

6.S965

Digital Systems Laboratory II

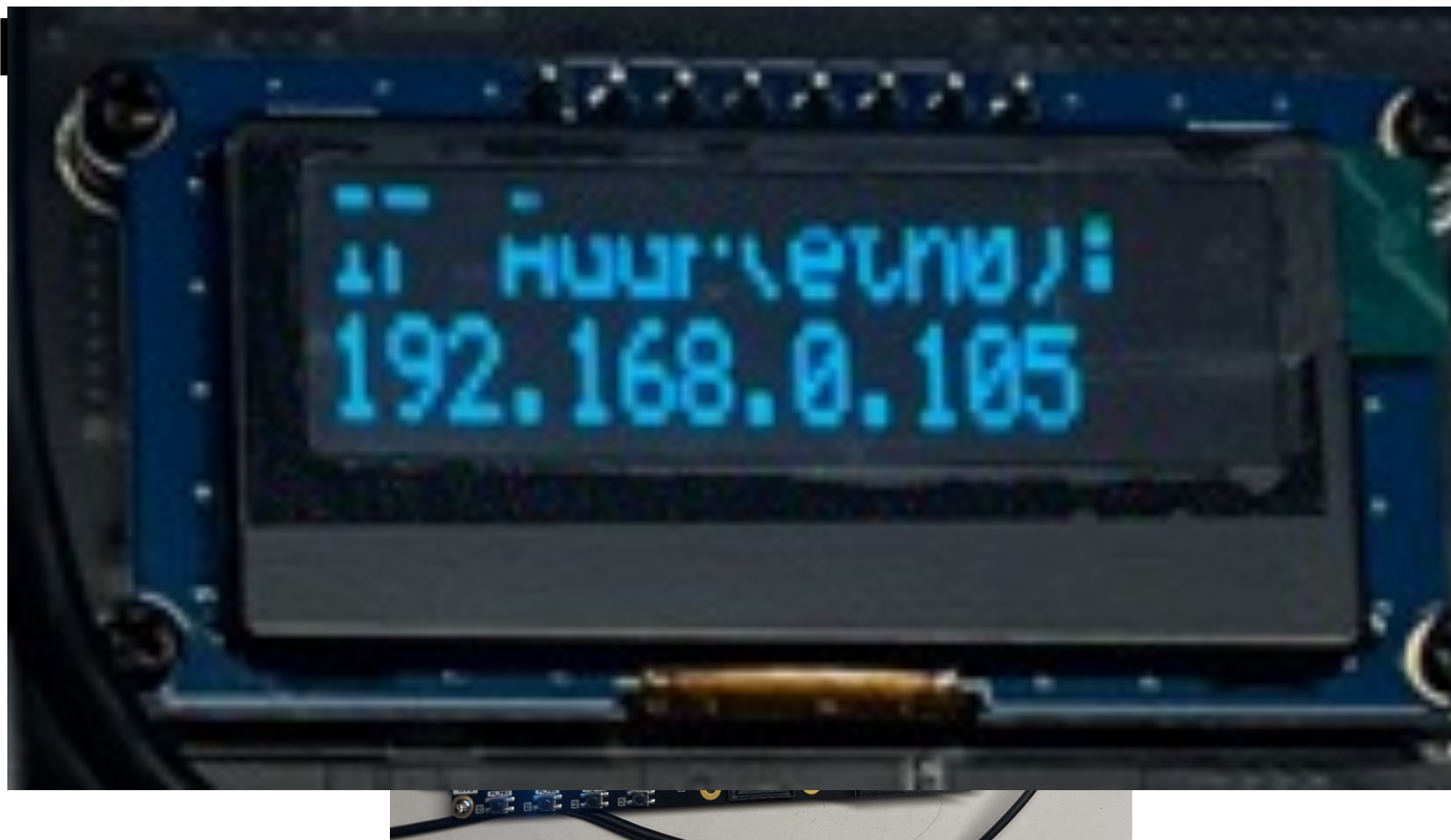
Lecture 11

RFSoc, Skid Buffers, Projects

Administrative

- Week 5 due on Friday
- Week 6 out on Friday:
 - Write, test Skid Buffer
 - Work with RFSoc
- Need to start thinking about projects

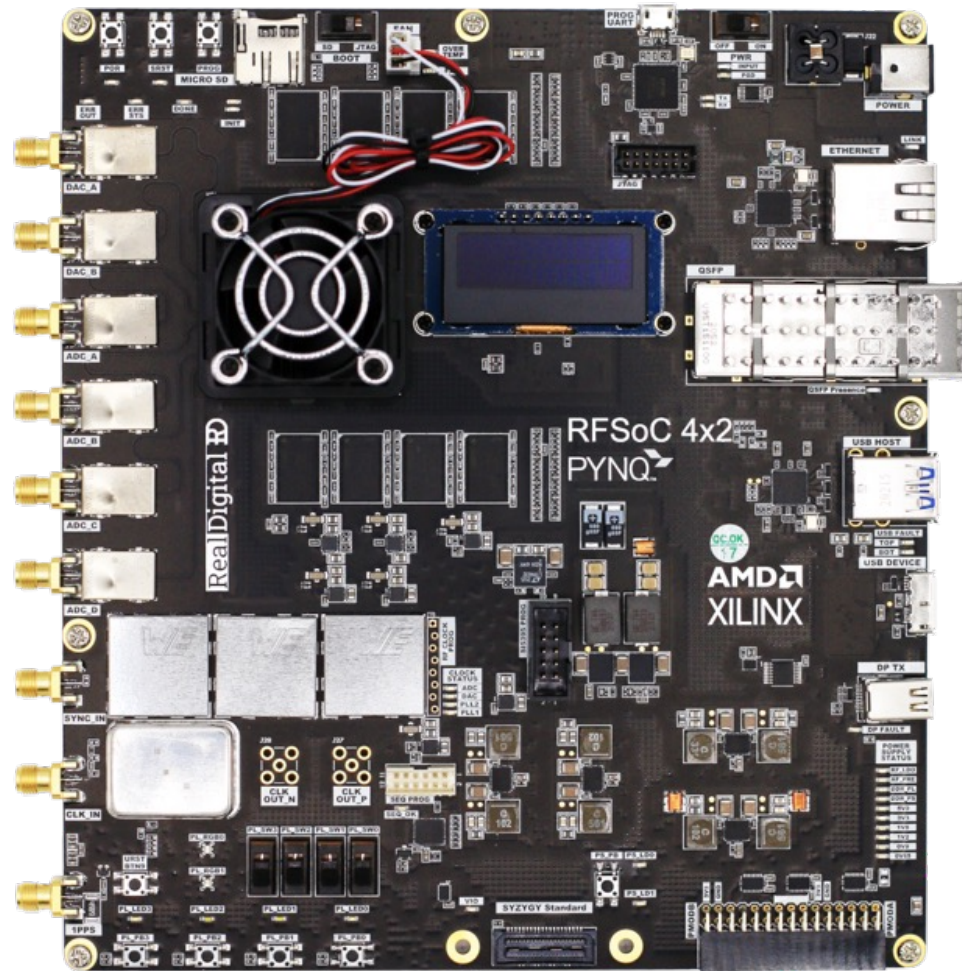
RFSoc



- For starters we'll just configure and run the ADC and LO and dump some data into memory to analyze

RFSoc

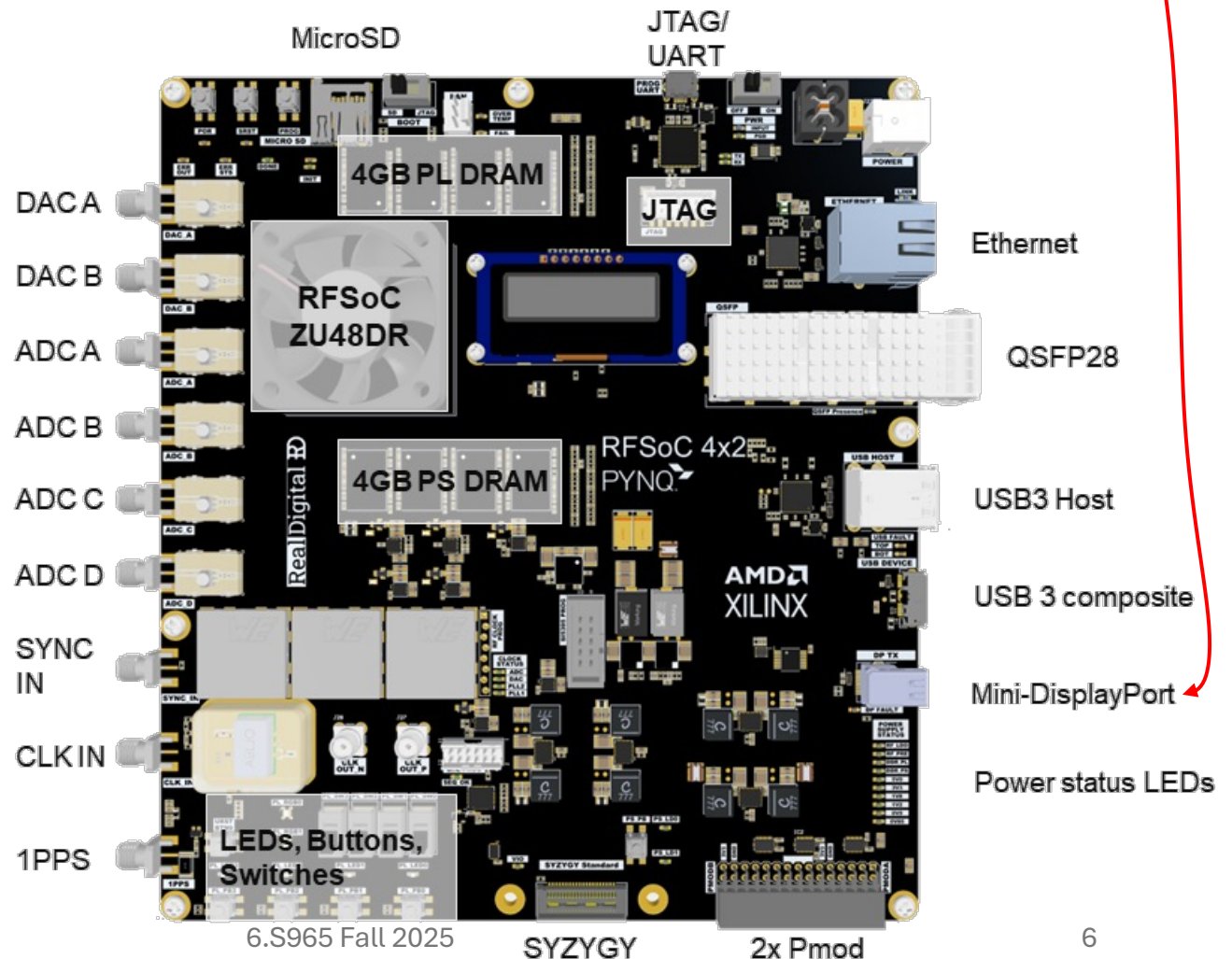
- It is basically the Pynq Z2 but with a lot more resources, the ADC, DAC and a lot less “fun” breakout parts like HDMI
- Chances are, outside of the ADC and DAC, most other peripherals have minimal support fyi.



We've tested the...

Oh there's a display port

- ADC
- DAC
- LEDs
- Switches
- Buttons
- Pmod pins
- DMA



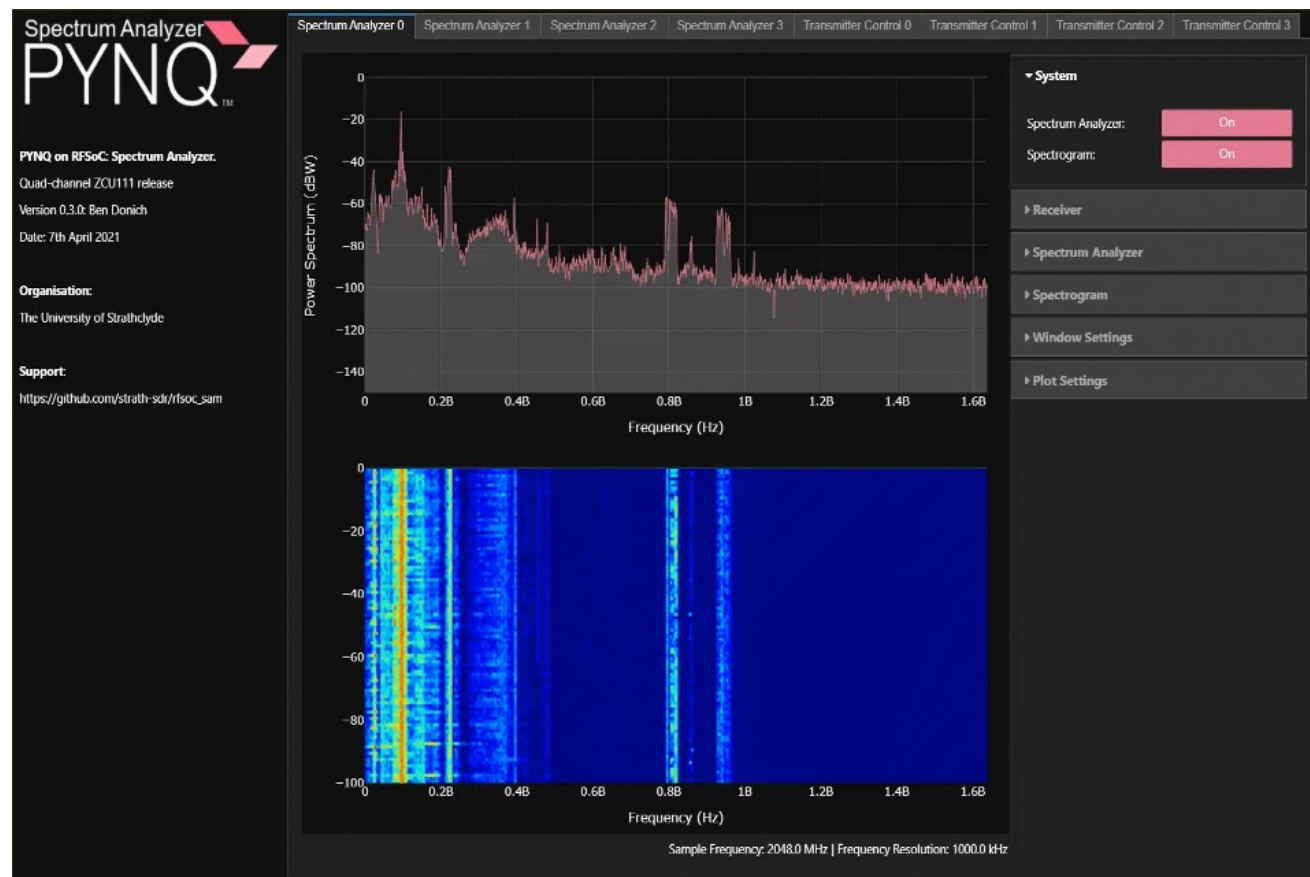
rfsoc.pynq.io

- Reference one:

The screenshot shows the website rfsoc.pynq.io in a web browser. The left sidebar contains a navigation menu with the following items: RFSOC-PYNQ, Main Menu, Home, RFSOC 4x2, Overview, Getting started guide, Resources, Overlays, Educational Resources, Tutorials, RFSOC Accessories, Support, Purchase, PYNQ support forum, FAQs, Legacy, and RFSOC 2x2. The main content area features a large banner for a new free eBook titled "Software Defined Radio Systems with Zynq® UltraScale+ RFSOC". The banner includes an image of the eBook cover and a Zynq UltraScale+ RFSOC development board. Below the banner, the text states: "This book introduces Zynq UltraScale+ RFSOC, a technology that brings real, single-chip, Software Defined Radio (SDR) to the marketplace. The book is accompanied by Jupyter Notebooks that can be run on your RFSOC-PYNQ enabled board, illustrating key concepts including *sampling and quantisation*, *filter design*, *Fourier's theorem* and *FFTs*, *pulse shaping*, *QAM*, *frequency planning*, *Forward-Error-Correction*, and *OFDM*." It also provides a link to the book website www.rfsocbook.com to download a free copy of the eBook and details on how to purchase hard copies. Below this, there is a section titled "New RFSOC-PYNQ release" which states: "The latest RFSOC-PYNQ 3.0 release adds supports for the ZCU208 alongside the existing support for the RFSOC 4x2, RFSOC 2x2, and ZCU111. To download the latest PYNQ image for your board, see [PYNQ.io board images](#)." The bottom section is titled "RFSOC-PYNQ" and describes it as an extension to PYNQ bringing support for the AMD-Xilinx Zynq RFSOC family of devices. It mentions that RFSOC created a new class of integrated circuit architecture for the communications and instrumentation markets, combining high-accuracy ADCs and DACs operating at Giga samples per second (GSPS), with programmable heterogeneous compute engines. It also states that RFSOC-PYNQ provides Python APIs, libraries and drivers for the RFSOC, example overlays and designs, tutorials and other resources for RFSOC users. The AMD Xilinx Zynq RFSOC logo is displayed in the bottom right corner. At the bottom left, there is a link to "View on GitHub".

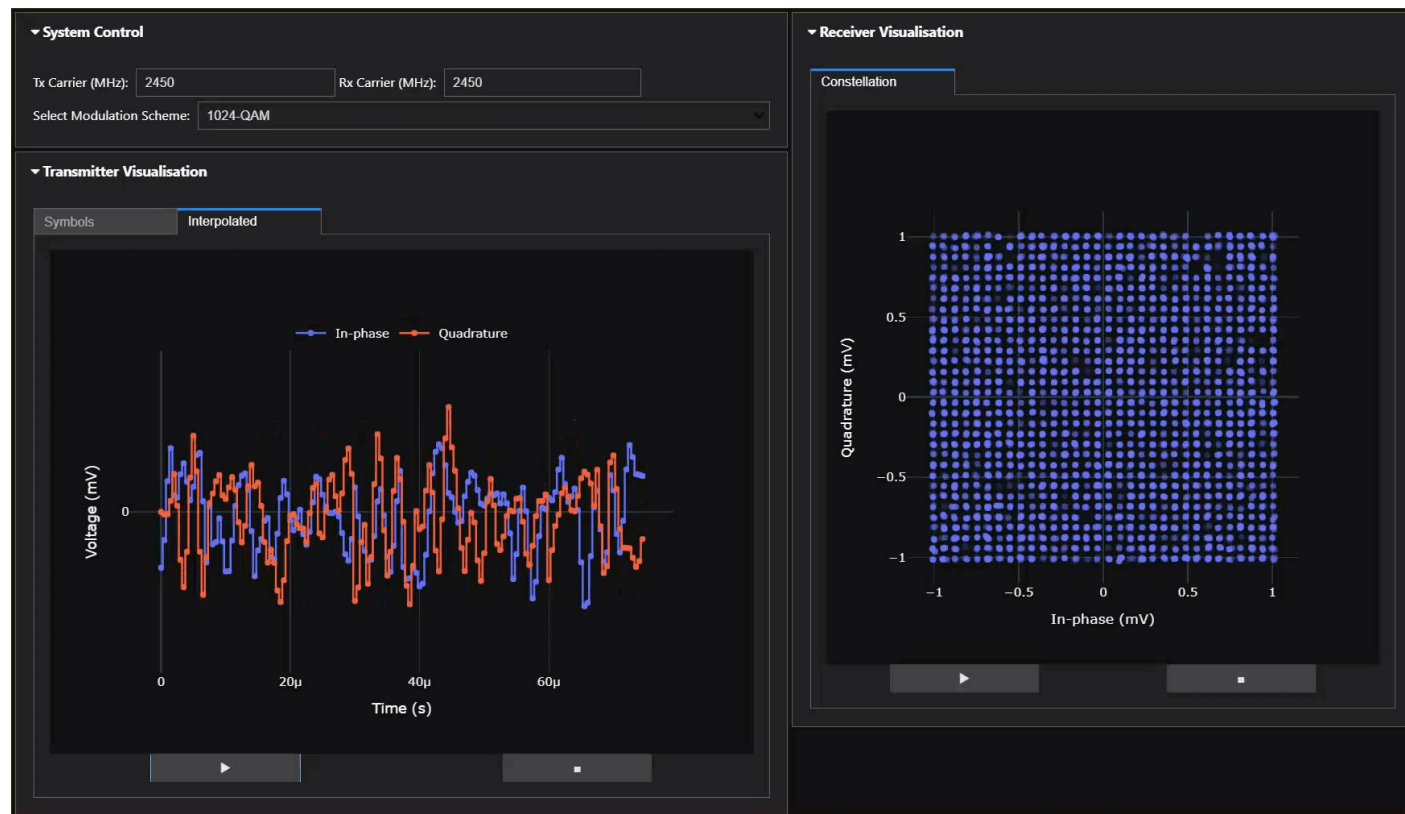
Has some interesting overlays

- Regular Old Spectrum Analysis



Has some interesting overlays

- 1024-QAM demo



Downsides

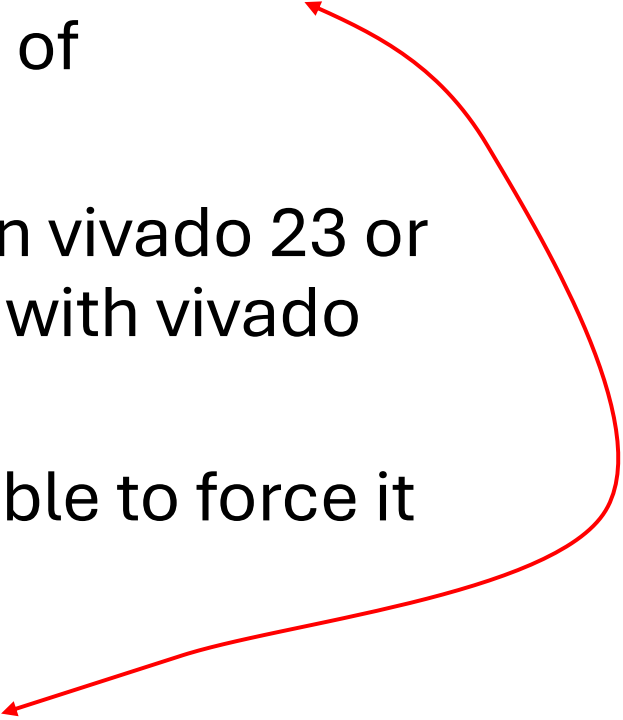
- All the code and widgets are found here:
 - https://github.com/strath-sdr/rfsoc_studio/tree/master
- All of their stuff was built with old Vivado (2022 or before). It does not build on 2023 or 2024, and they don't seem in any hurry to do anything about that.



This repository contains the source code and build scripts for the RFSoc-PYNQ base design and SD card images. The design files in this repository are compatible with Xilinx Vivado 2022.1, and PYNQ v3.0.0 and later.

Currently, the ZCU111, ZCU208, RFSoc4x2 and RFSoc2x2 platforms are supported.

You can run their notebooks and there's some interesting demos

- Most of which are built off of this “base.bit” overlay they wrote which has a lot of functionality in it.
 - But you can't open it or rebuild it in vivado 23 or 24 (and even when I tried early on with vivado 22.2 it broke)
 - Haven't tried 2025. We might be able to force it with some work.
- 

Base.bit is a massive project they built with most functionality broken out (not good for specialized things though)

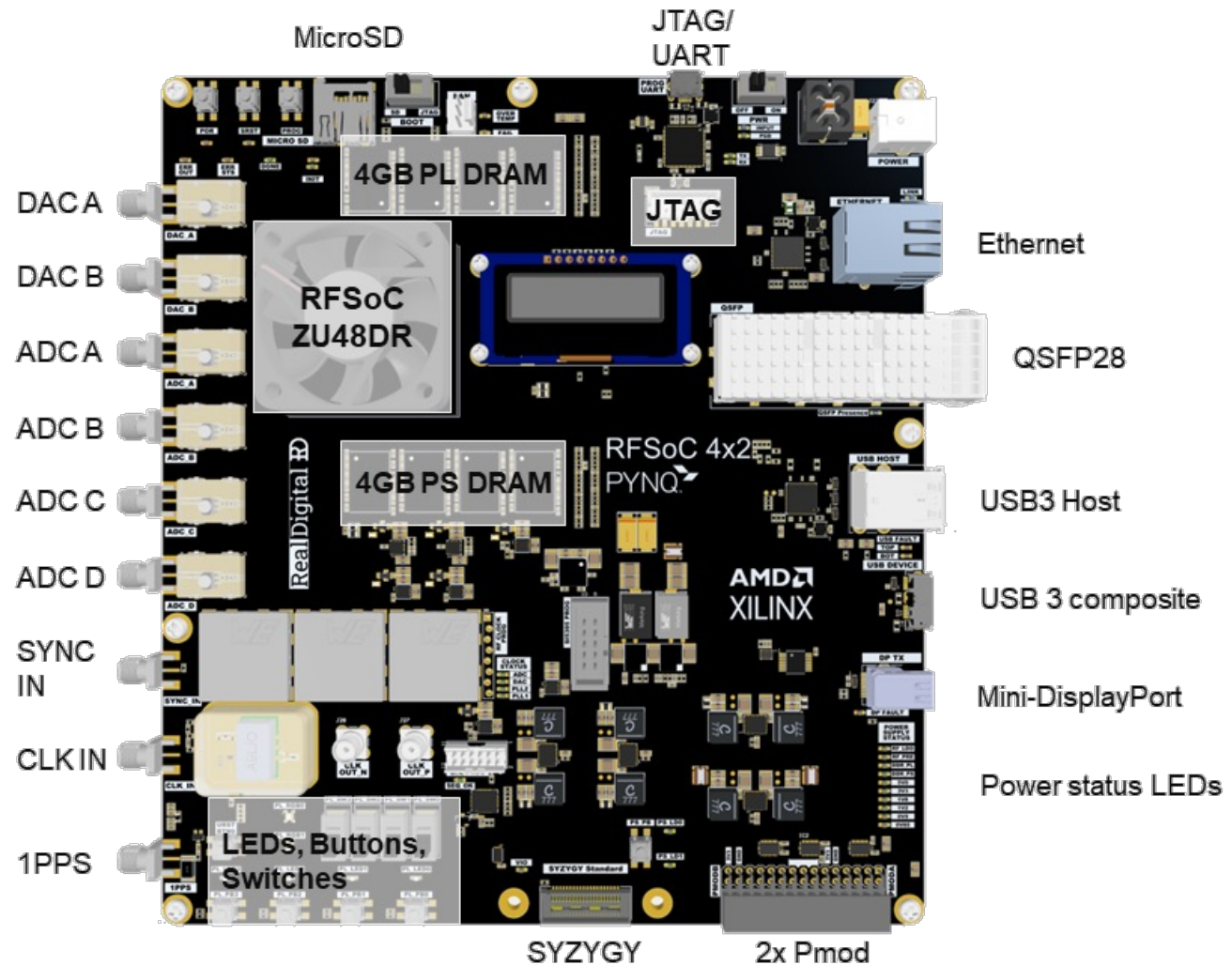
Good News

- Everything we've done on the cheaper Pynq board, including DMA transfers relatively quickly right over.
- Things do seem to work with the Pynq...it isn't that it is unsupported in my experience, it is just there is nothing out there to show you how for specific things.
- There just aren't many walkthroughs about how to actually build something on the RFSoc out there.

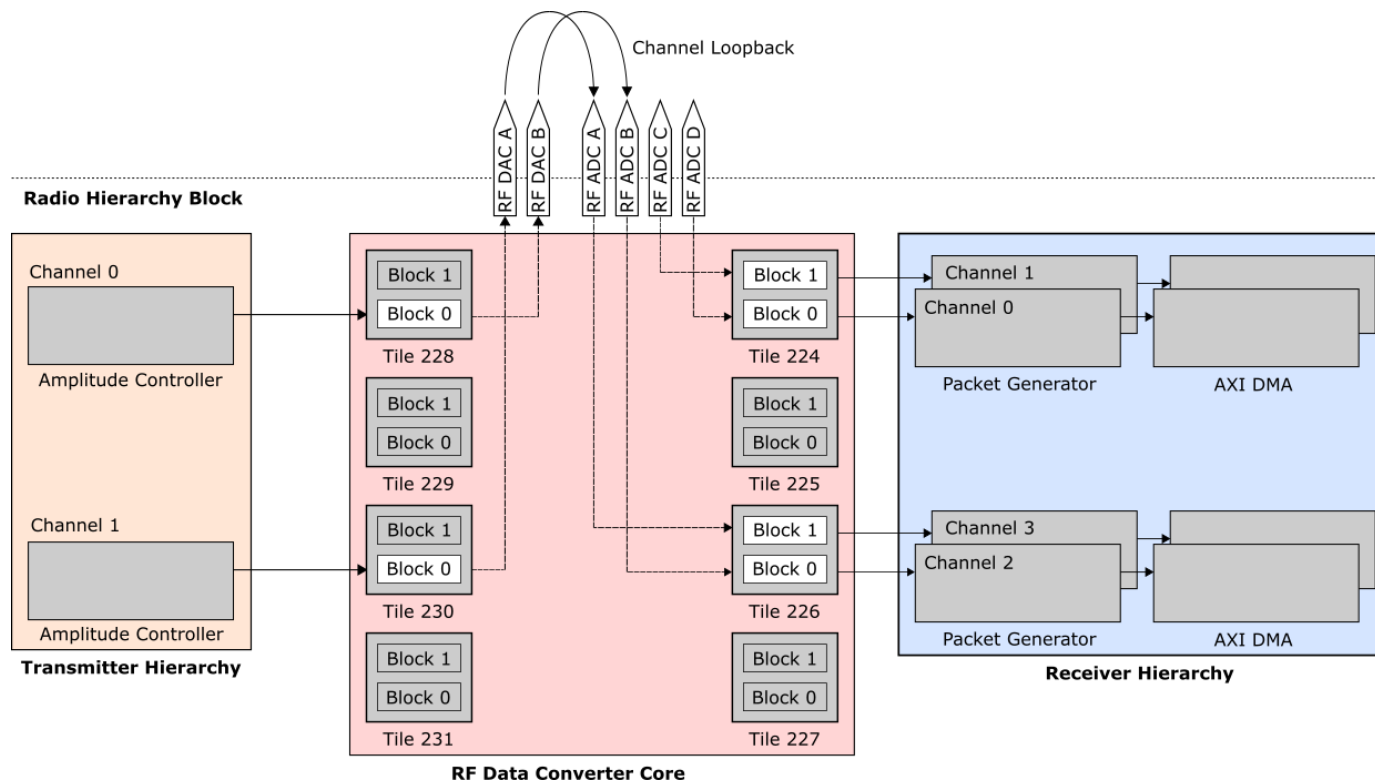
The RFSoc's *raison d'être*

- Yes yes it has crazy amounts of DSP blocks and logic, but it is all in service to...
- The ADC and the DAC

ADCs and DACs on left side



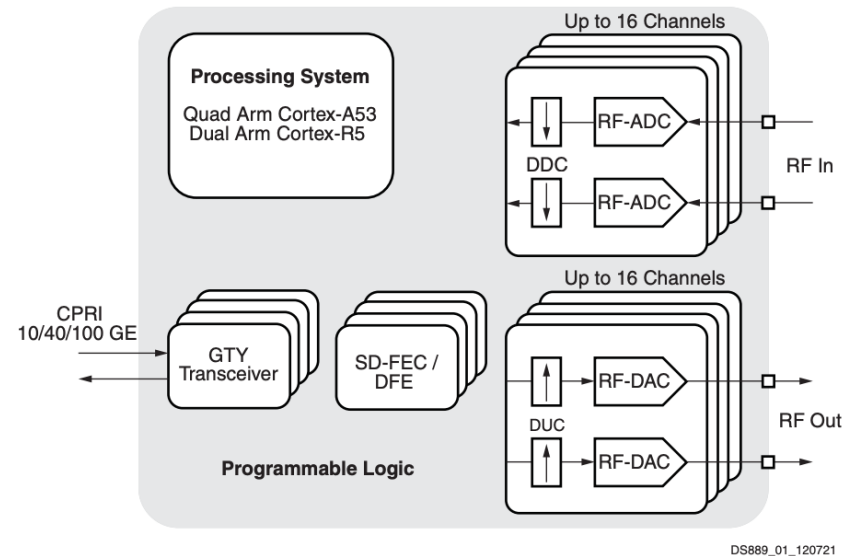
The RFSoc Board has DACs and ADCs in two blocks



RFSoc Capabilities

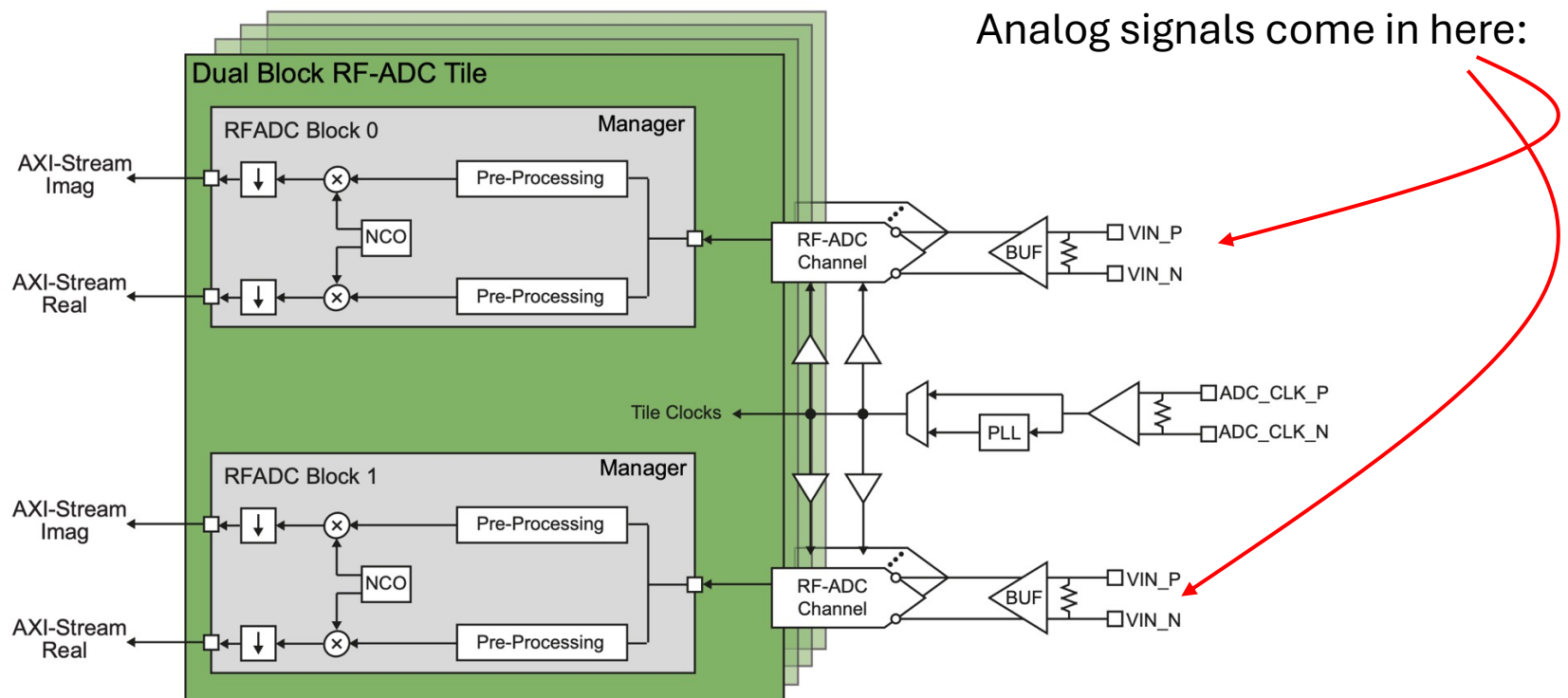
- DDC == “Digital Down Converter”
- DUC == “Digital Up Converter”

Key Components of the Zynq UltraScale+ RFSoc



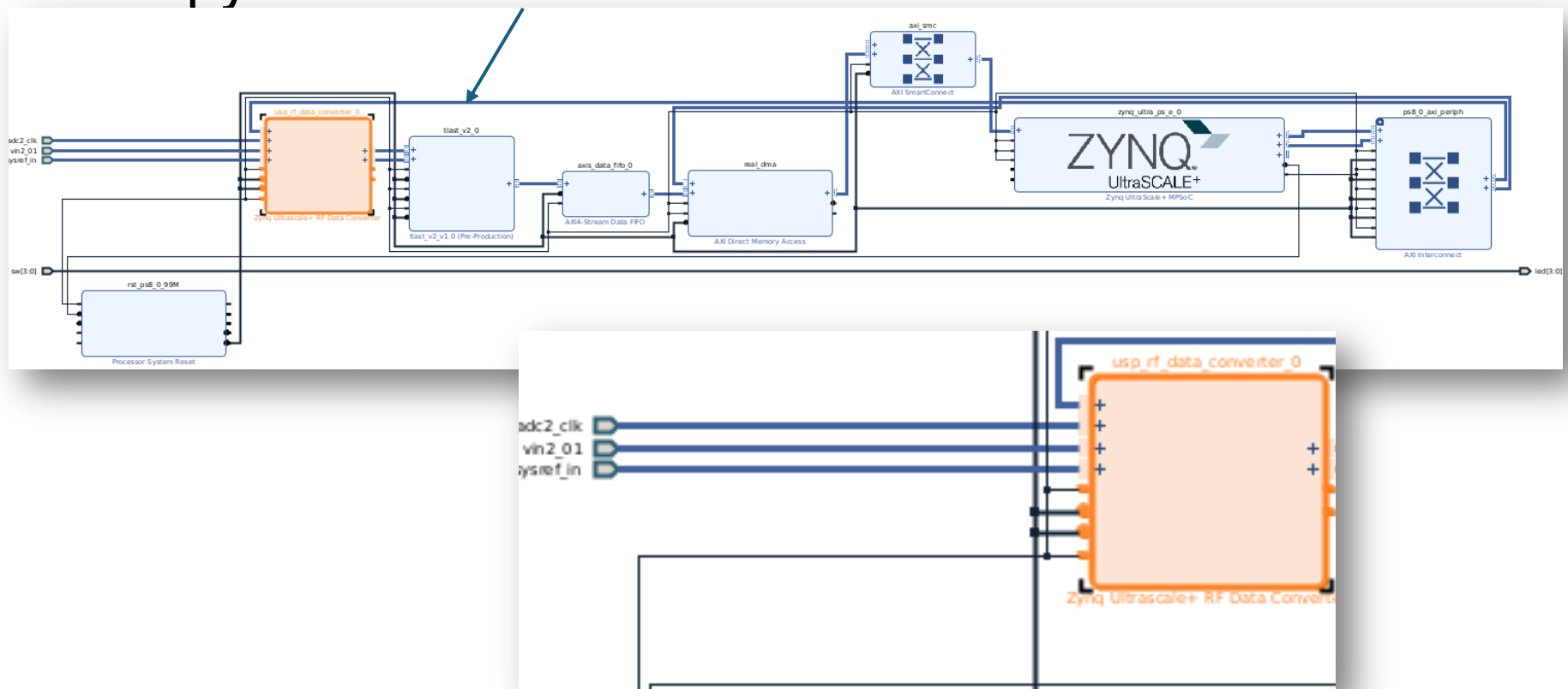
ADCs

- You pretty much need to use these in I/Q format to get most functionality

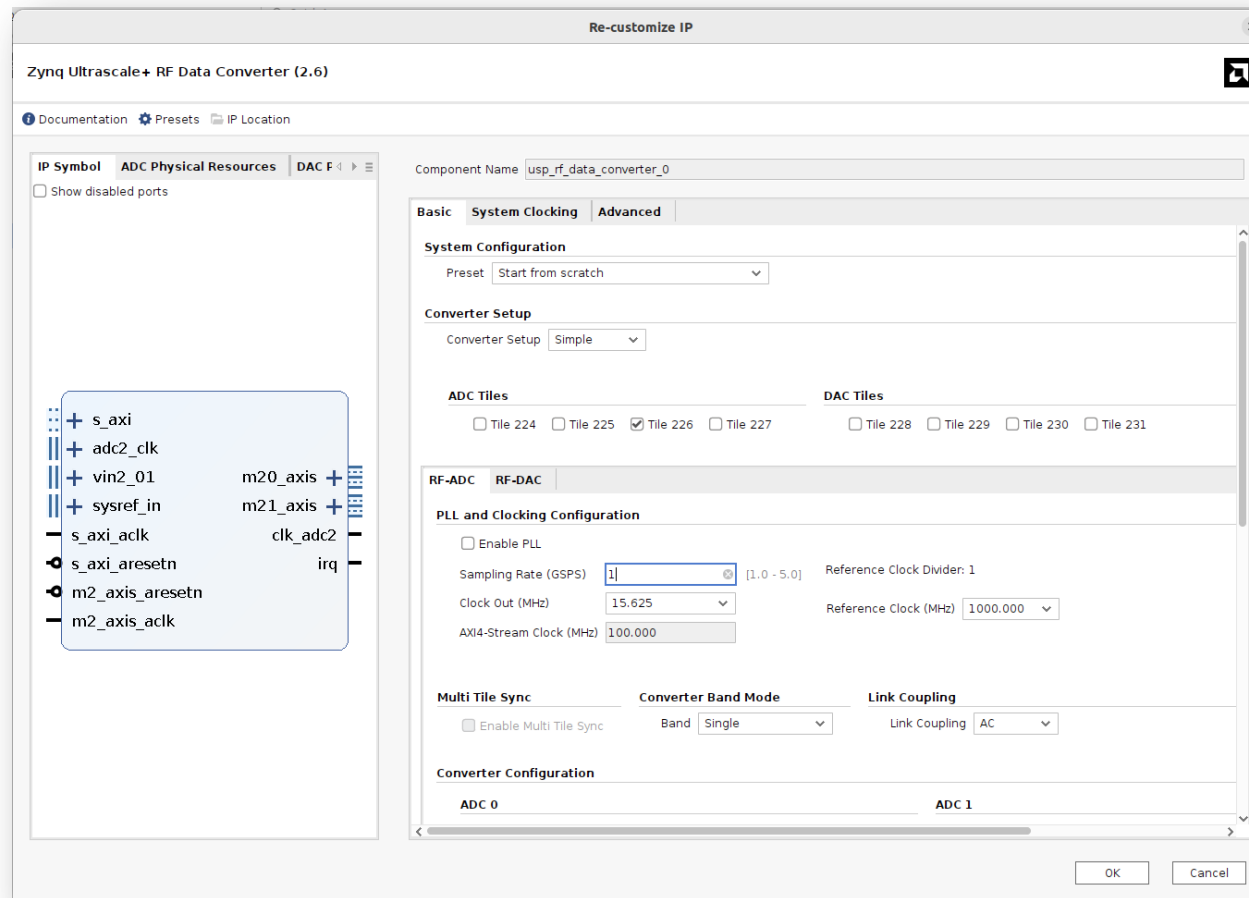


Week 6

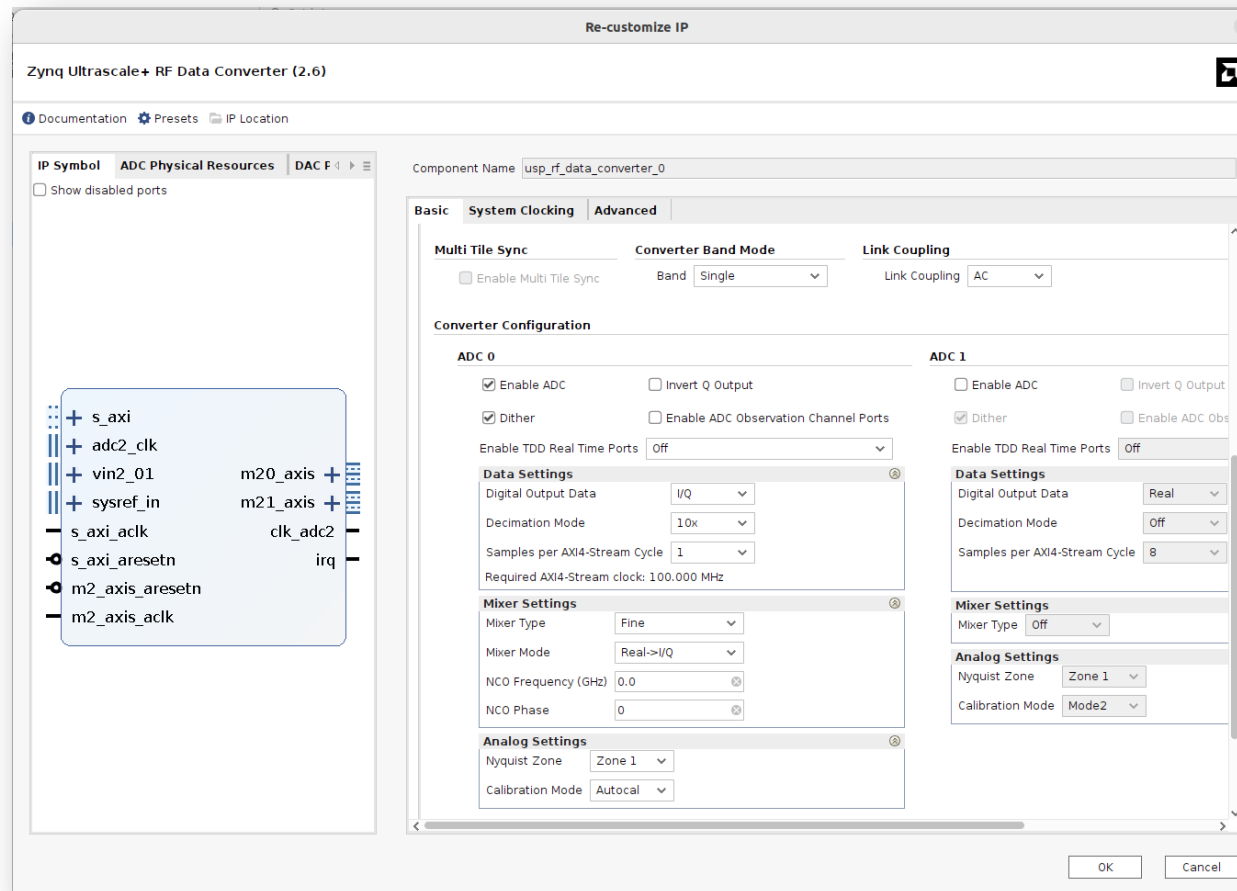
- Build an ADC pipeline and look at some signals in python RF DDC (ADC)



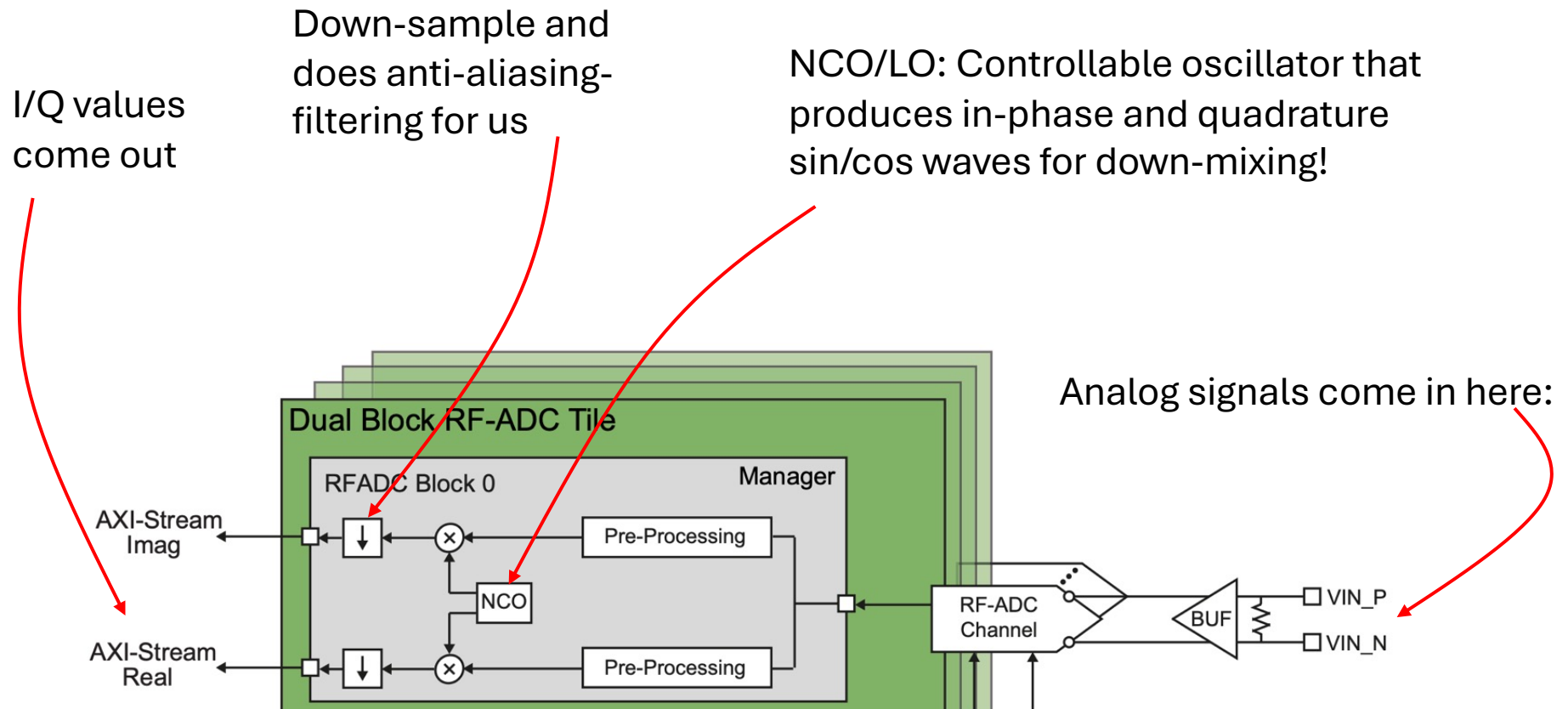
The RF Data Converter gets Configured...



The RF Data Converter gets Configured...



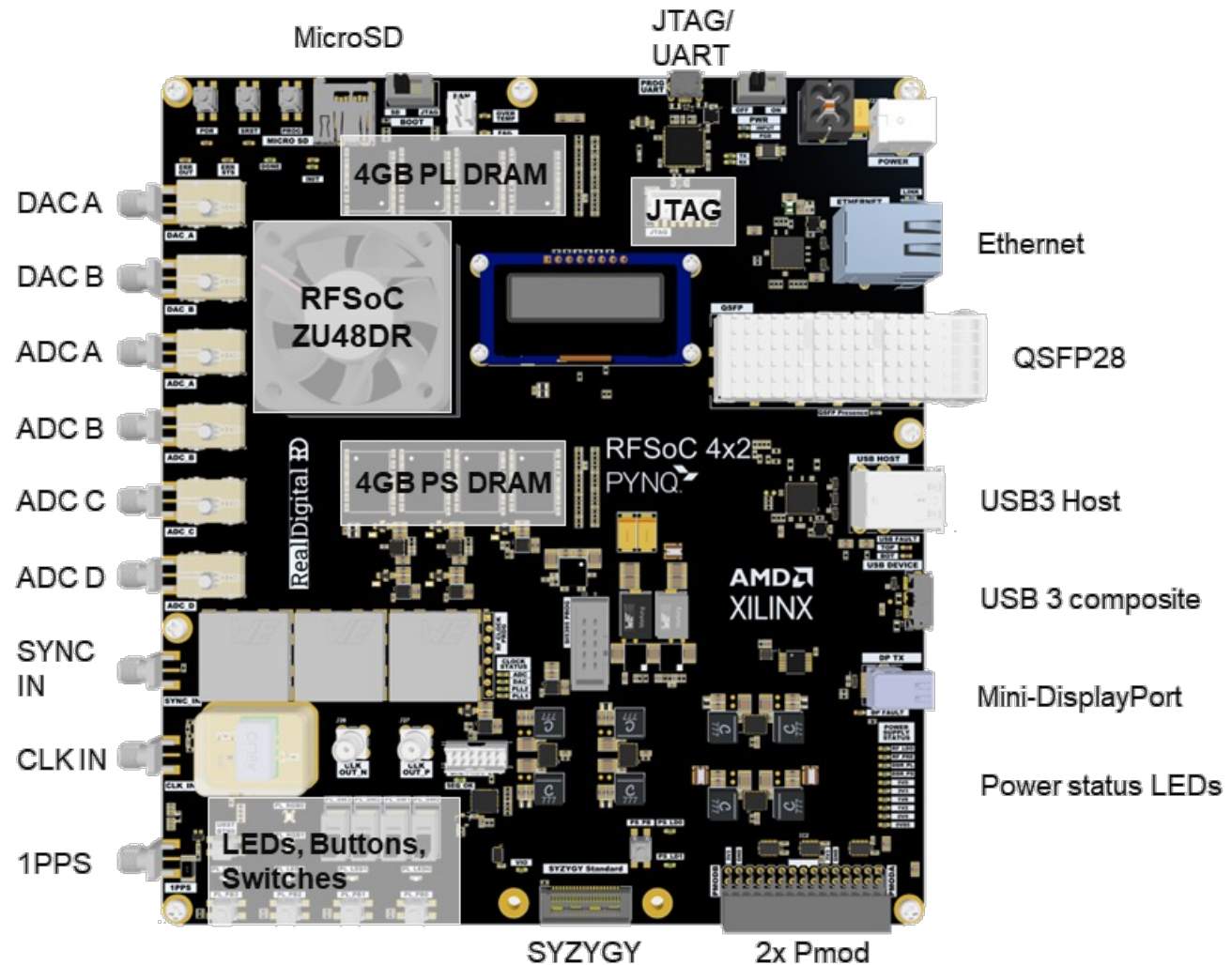
The RFADC Block



Can control mixer frequency from software

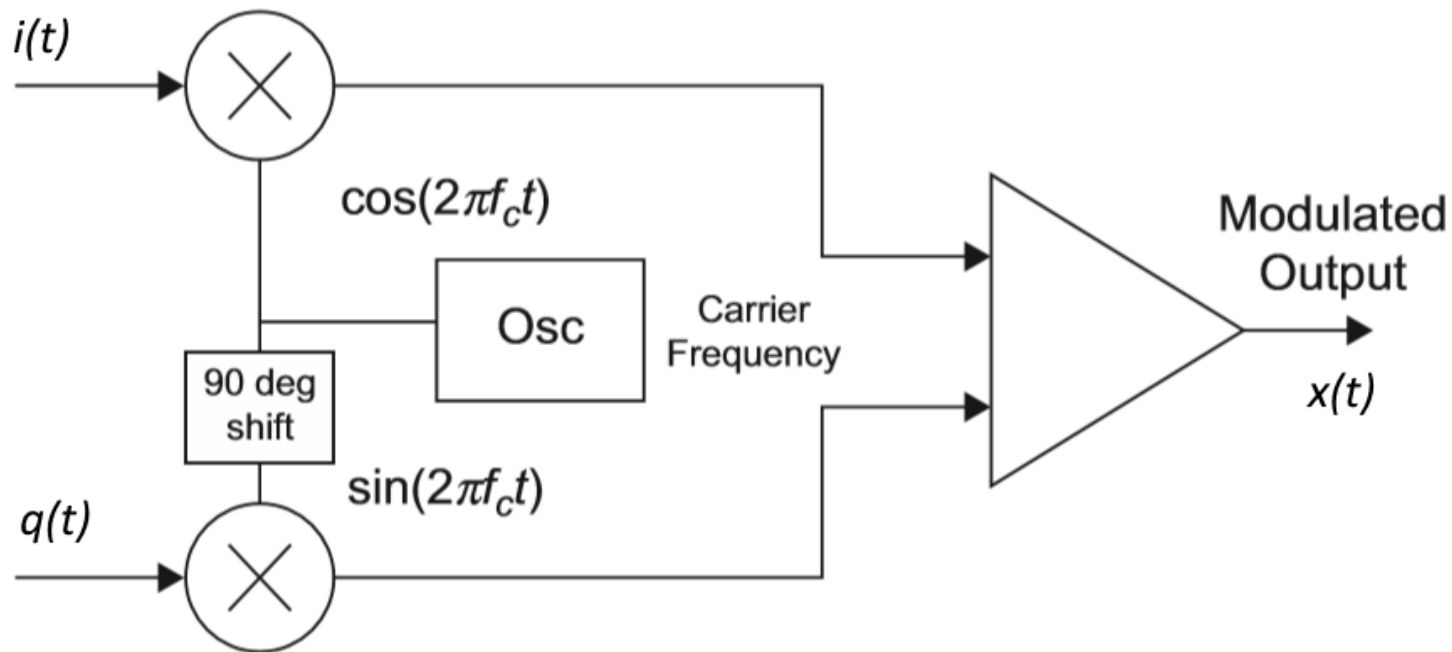
```
adc_block.MixerSettings['Freq']= 85  
adc_block.UpdateEvent(xrfdc.EVENT_MIXER) #every time setting is changed, must call this.
```

ADCs and DACs on left side



This is a Digital Upconverter (DUC)

- Specify complex values over time



<https://www.edn.com/quadrature-modulation-the-signal-behind-digital-communications/>

DACs

- Basically work the opposite of DDC

Analog signals get sent out here:

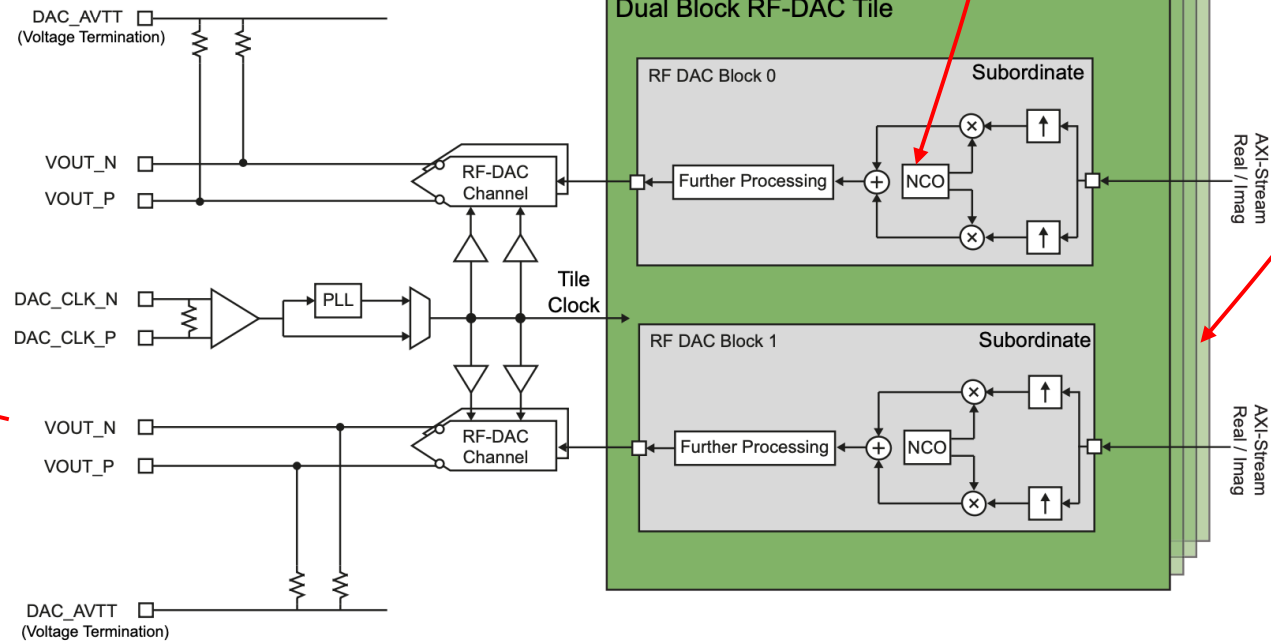


Figure 3.11: RF-DAC hierarchy for the ZU48DR RFSoc device [90].

RFSoc Gen 3

- Specs...

RF-ADC Features

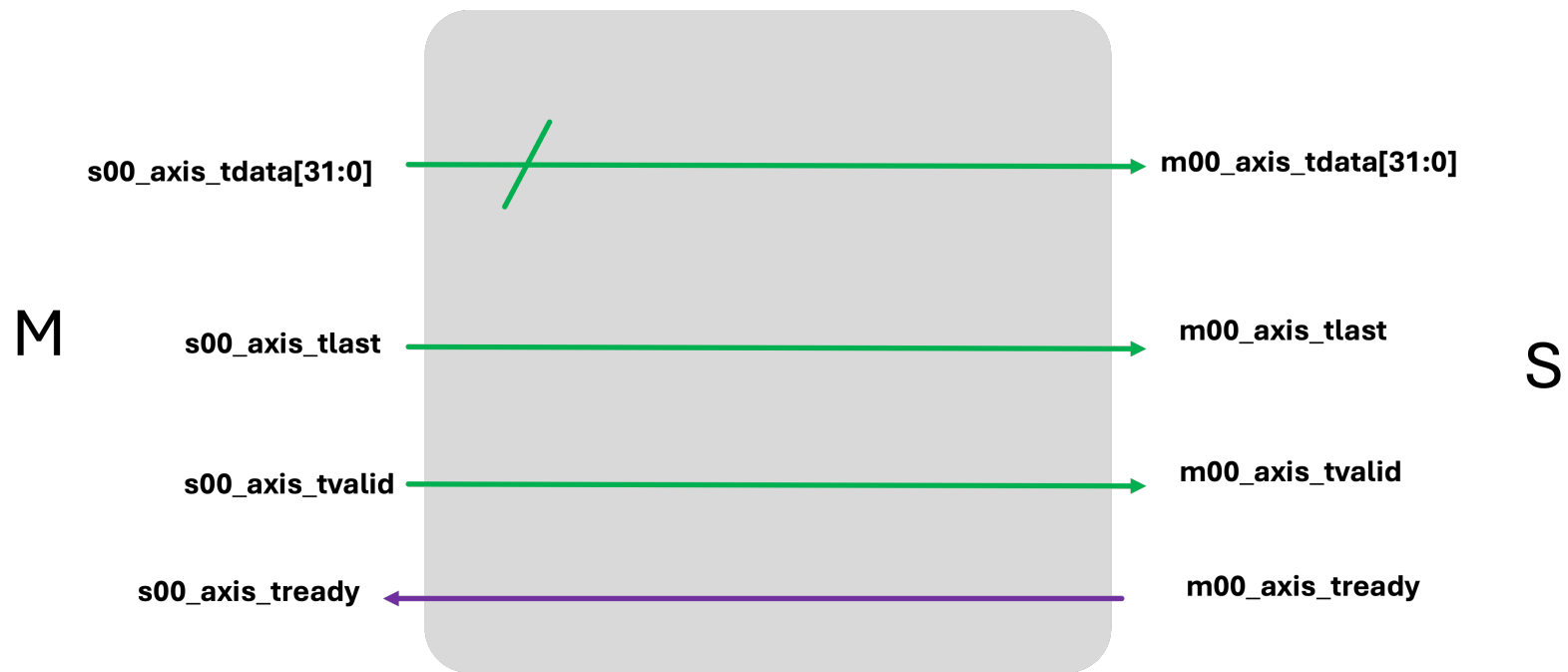
- Tile configuration
 - Four or two RF-ADCs and one PLL per tile
 - Gen 1/Gen 2: 12-bit RF-ADC resolution, with 16-bit digital signal processing datapath; each 12-bit data stream is MSB-aligned to 16-bit samples at the output of the RF-ADC core before passing to the DDC block
 - Gen 3/DFE: 14-bit RF-ADC resolution, with 16-bit digital signal processing datapath; each 14-bit data stream is MSB-aligned to 16-bit samples at the output of the RF-ADC core before passing to the DDC block
 - Implemented as either four channels (Quad) or two channels (Dual) (the sampling rate is device dependent; for the actual sampling rate specifications, see the *Zynq UltraScale+ RFSoc Data Sheet: DC and AC Switching Characteristics (DS926)*)
- Decimation filters
 - Gen 1/Gen 2: 1x (bypass filter), 2x, 4x, 8x
 - Gen 3/DFE: 1x (bypass filter), 2x, 3x, 4x, 5x, 6x, 8x, 10x, 12x, 16x, 20x, 24x, 40x
 - 80% of Nyquist bandwidth, 89 dB stop-band attenuation
- Digital Complex Mixers
 - Full complex mixers support real or **I/Q** inputs from the RF-ADC
 - 48-bit Numeric Controlled Oscillator (NCO) per RF-ADC
 - Fixed $F_s/4$, $F_s/2$ low power frequency mixing mode, where F_s is the sample frequency
 - **I/Q** and real input signals supported
- Single/multi-band flexibility
 - 2x bands per RF-ADC pair
 - 4x bands per Quad RF-ADC tile
 - Can be configured for real or **I/Q** inputs
- Full bandwidth of the RF-ADC can be accessed in bypass mode
- Input signal amplitude threshold: Two programmable threshold flags per RF-ADC
- Built-in digital correction for external analog quadrature modulators:
 - Supports gain, phase, and offset correction for an **I/Q** input pair (two RF-ADCs)
- SYSREF input signal for multi-channel synchronization

Skid Buffer

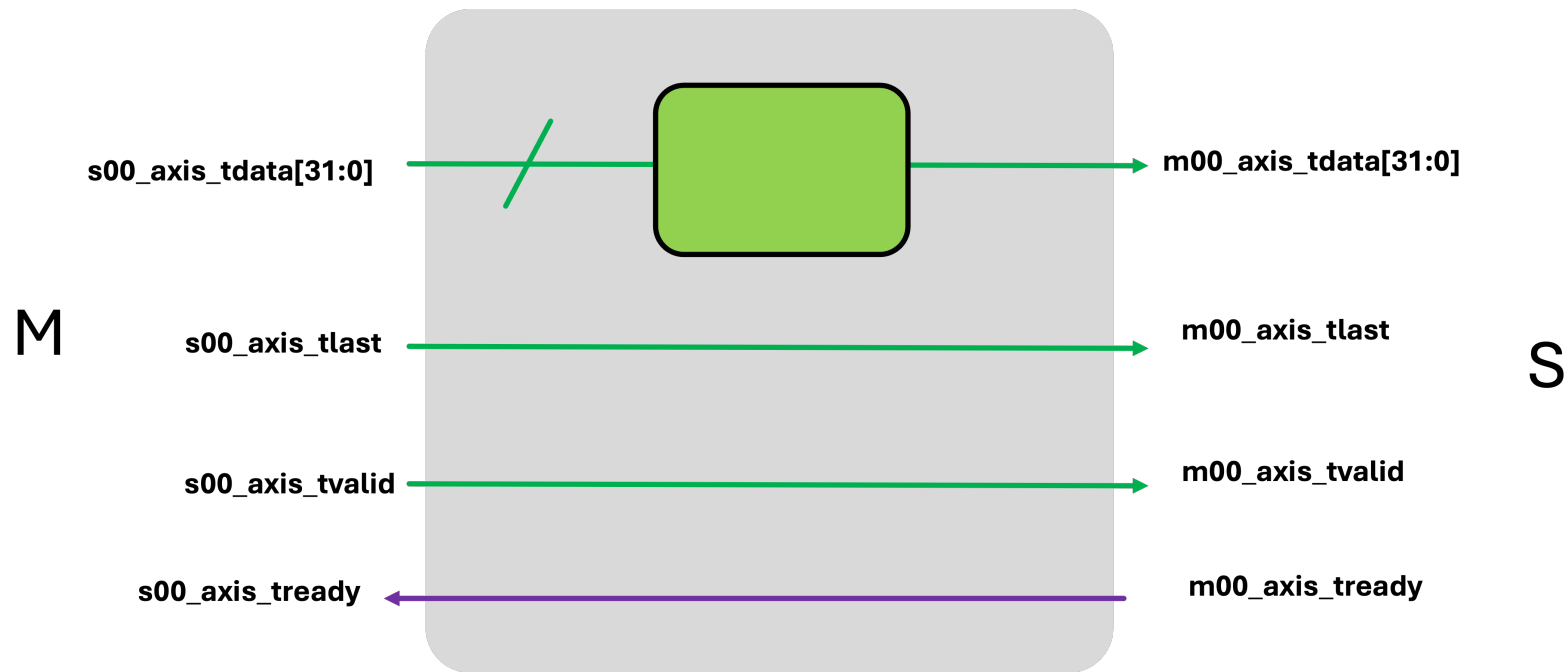
Motivation for Skid Buffer

- We've been writing AXIS modules for a few weeks. One issue we've kinda dealt with and then ignored some was TREADY propagation

Original AXI Wire

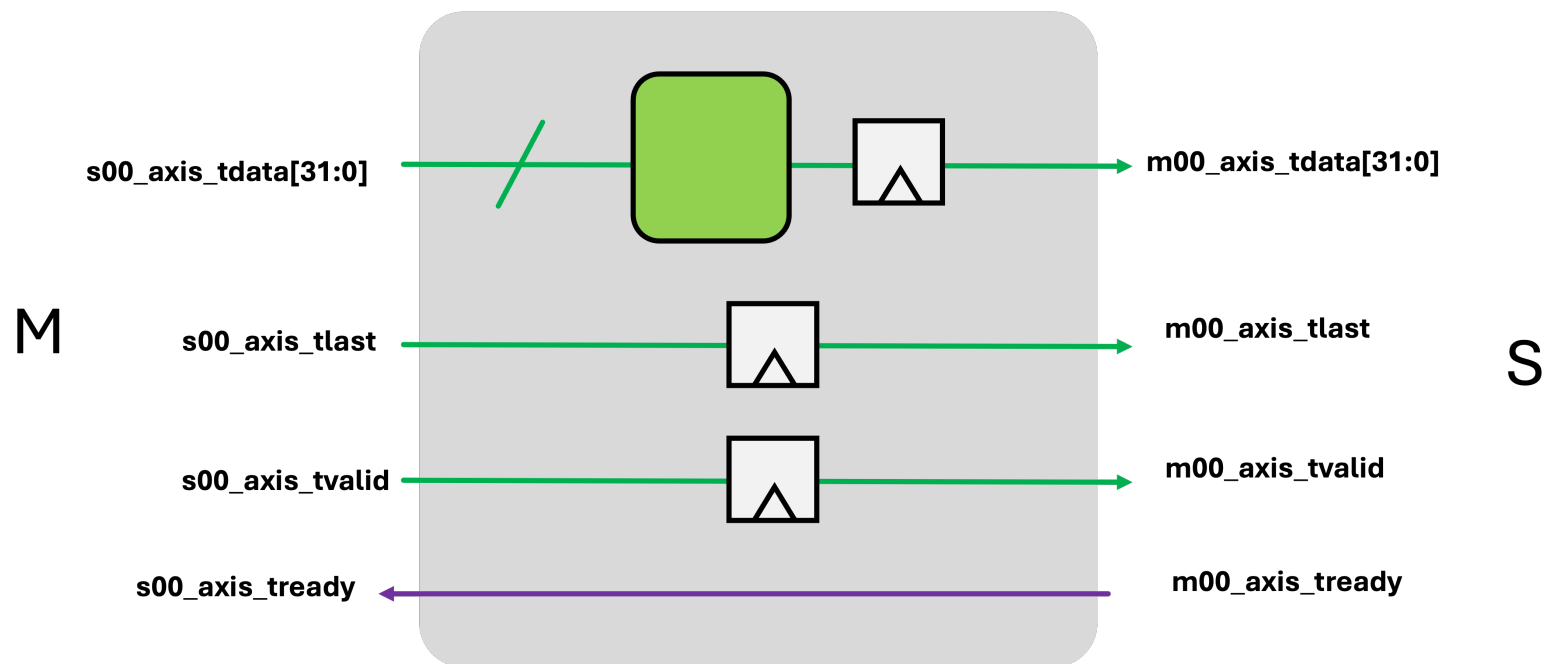


As soon as you want to do stuff, though

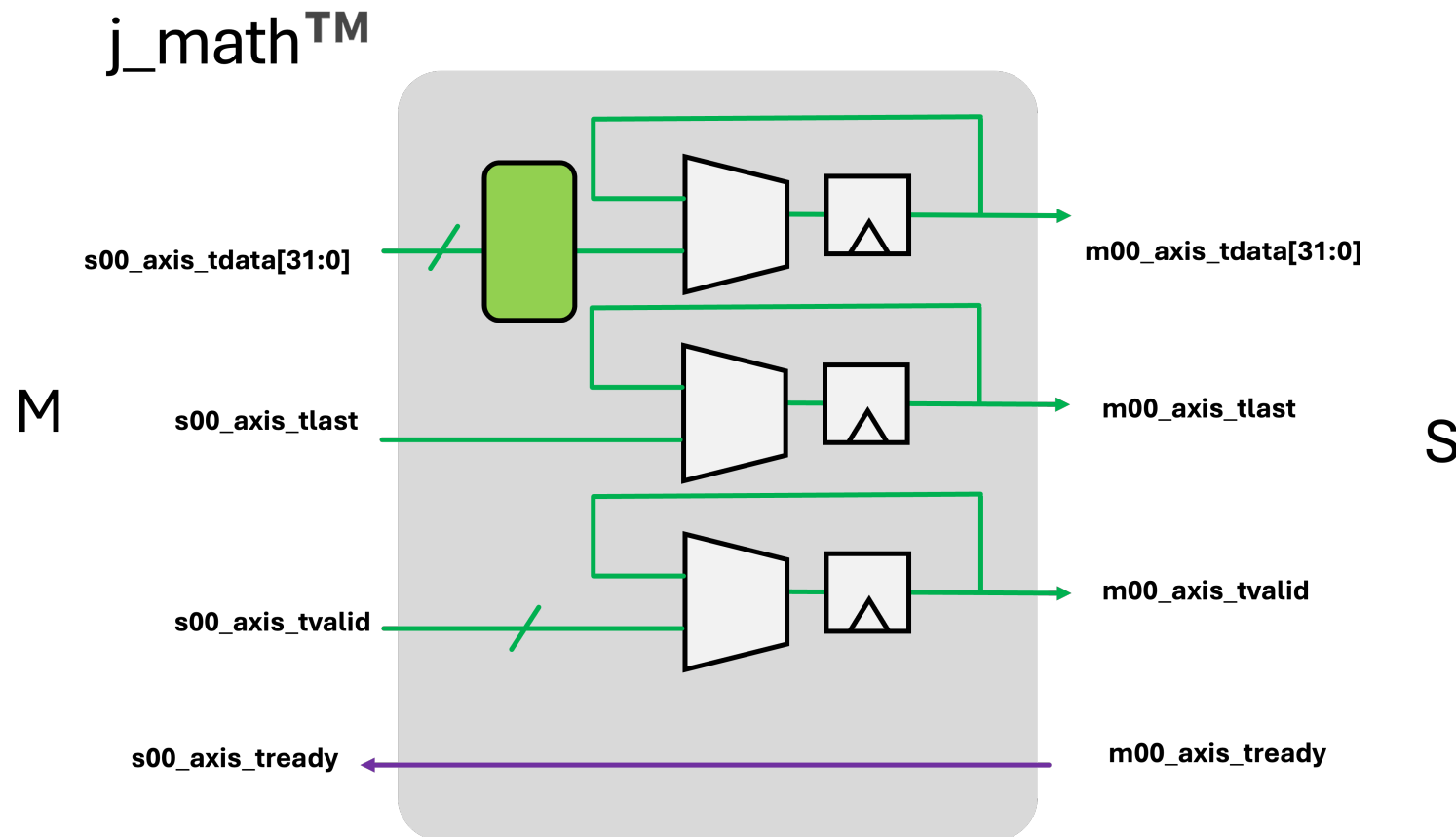


You likely need to register what you're doing

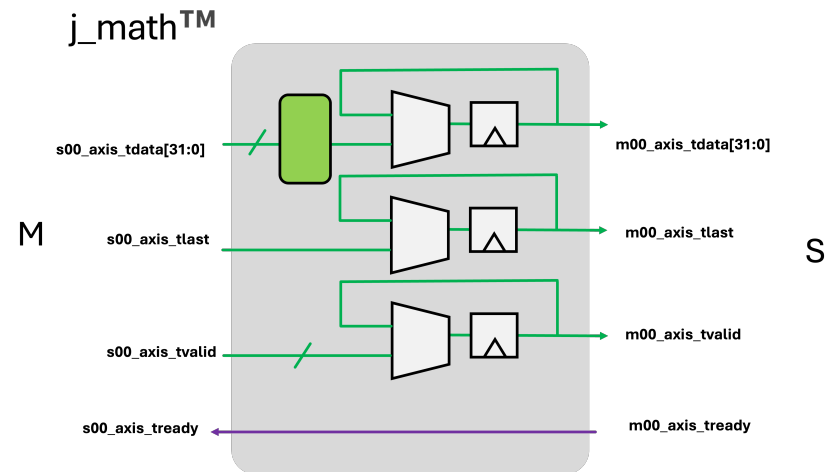
- And this brings a lot of baggage with it. The module is now stateful.



And it can get more complicated as we saw with j_math, etc...

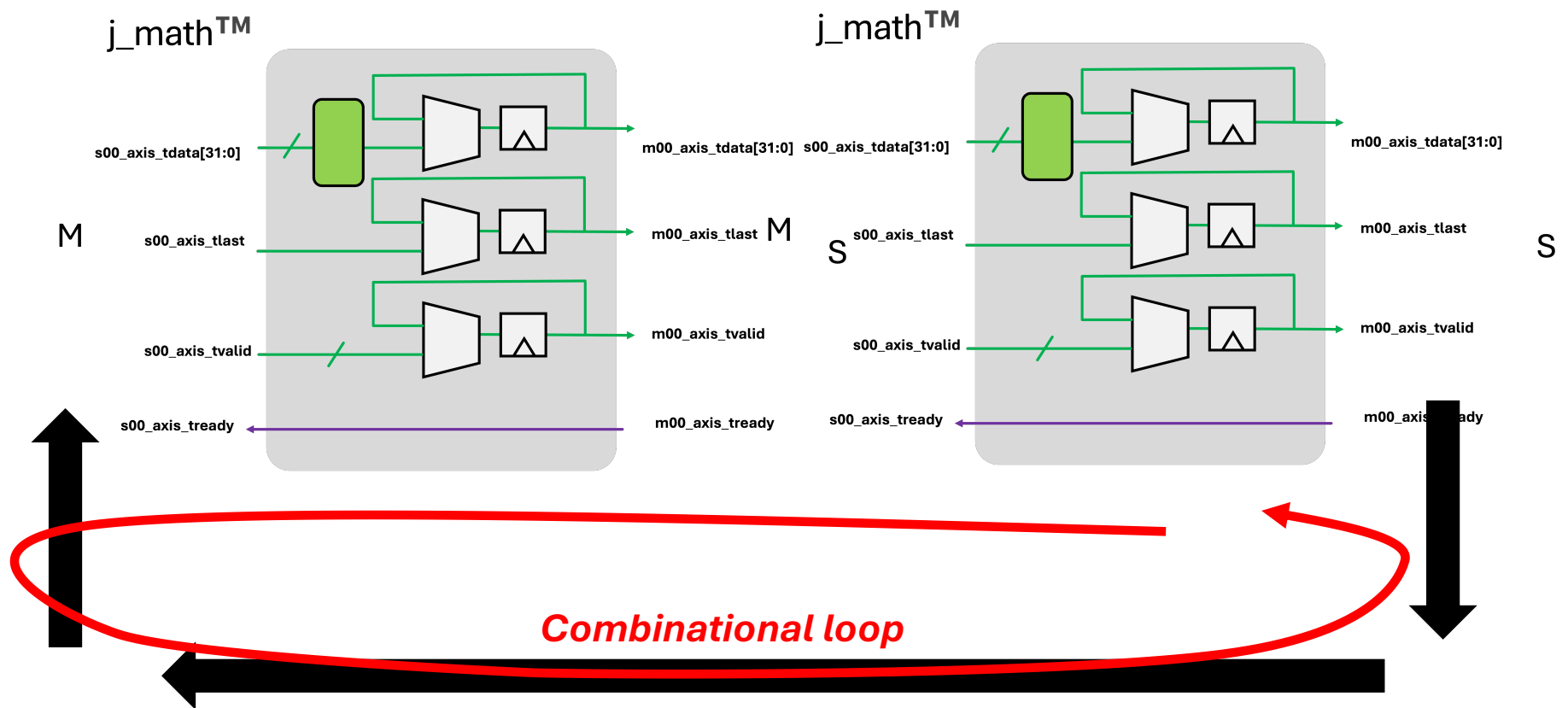


But One Issue



- We'd ideally like to tell the entire pipeline to stop all at once to avoid car-crash like build-ups because of the state.
- Do this with an immediately conveyed combinational path...
- Stacking lots of modules together could run into issues with long combinational paths making timing hard to meet.

Add into a feedback path or something...



Go to the Scripture (AXI Spec)

A3.2.1 Handshake process

All five transaction channels use the same **VALID/READY** handshake process to transfer address, data, and control information. This two-way flow control mechanism means both the master and slave can control the rate at which the information moves between master and slave. The *source* generates the **VALID** signal to indicate when the address, data or control information is available. The *destination* generates the **READY** signal to indicate that it can accept the information. Transfer occurs only when *both* the **VALID** and **READY** signals are HIGH.

On master and slave interfaces there must be no combinatorial paths between input and output signals.

Figure A3-2 to Figure A3-4 on page A3-38 show examples of the handshake process.

In Figure A3-2, the source presents the address, data or control information after T1 and asserts the **VALID** signal. The destination asserts the **READY** signal after T2, and the source must keep its information stable until the transfer occurs at T3, when this assertion is recognized.

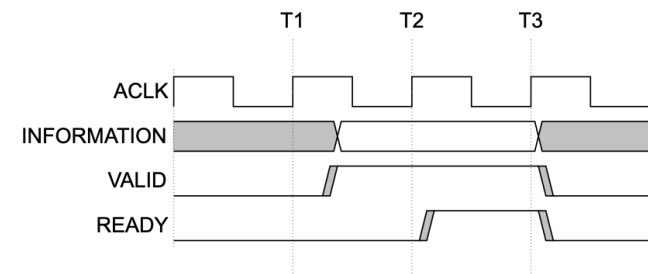


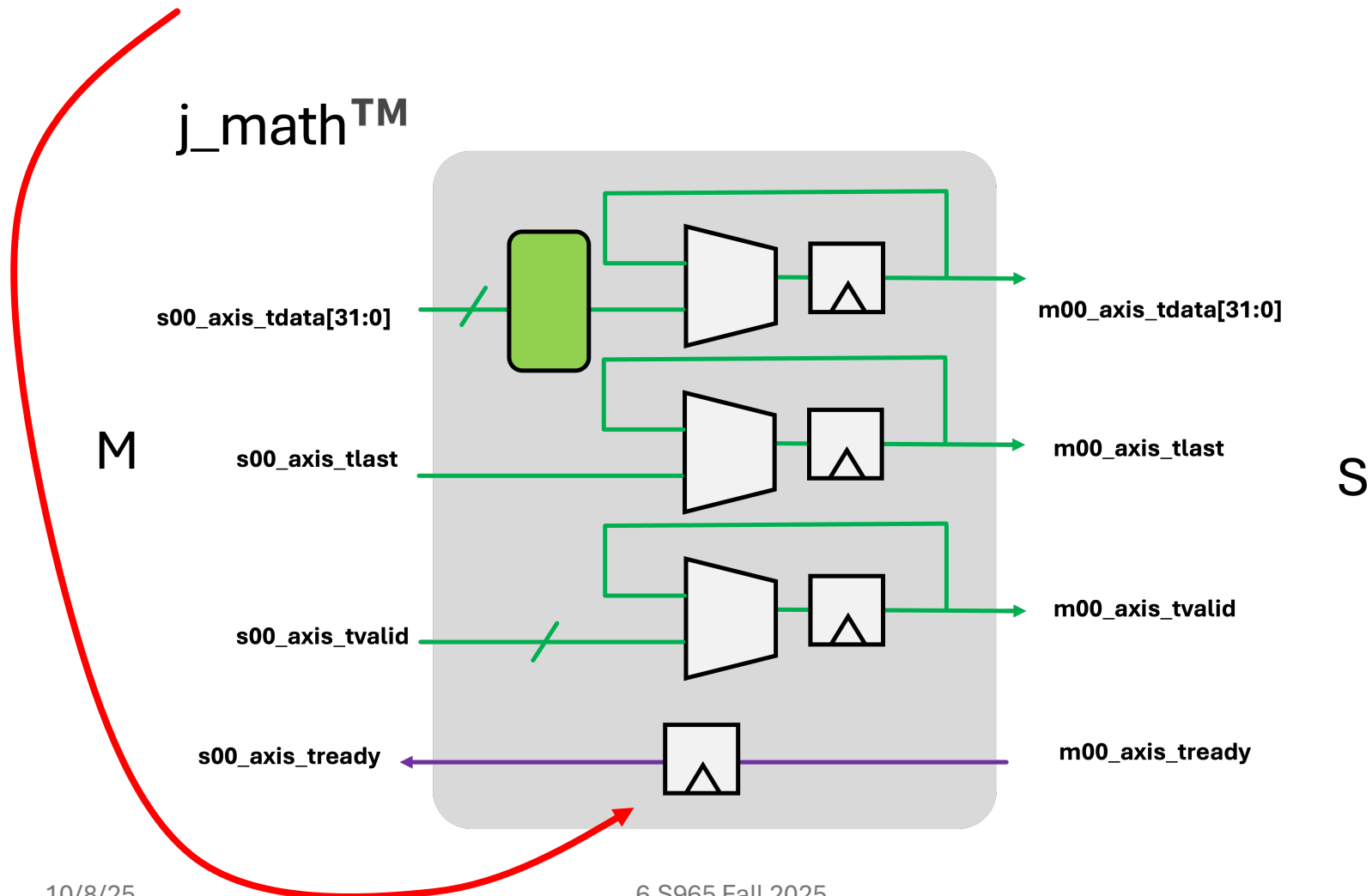
Figure A3-2 VALID before READY handshake

- Is it allowed or not? Not really sure.

Probably not a good idea in general

- Especially if you end up having AXI-stream devices cross clock domains and things (Which can happen)!

So probably want to add a register here.



Delaying TREADY

- Delaying the ability to convey a halt (via TREADY) to any upstream device means that there's a delay in stopping that data.
- It has to go somewhere/get absorbed somewhere
- Need a buffer/some sort of very short-form fifo
- You'll hear these called “skid buffers” or “Carloni Buffers”

<https://ptolemy.berkeley.edu/projects/embedded/research/hsc/class.F02/ee249/lectures/lipClass.pdf>

What is a Skid Buffer?

- A device that “eats”/temporarily holds data in the event of the data pipeline having to suddenly slam on the brakes.
- Therefore the system “skids” to a halt.
- In many ways, a skid buffer is the smallest possible FIFO

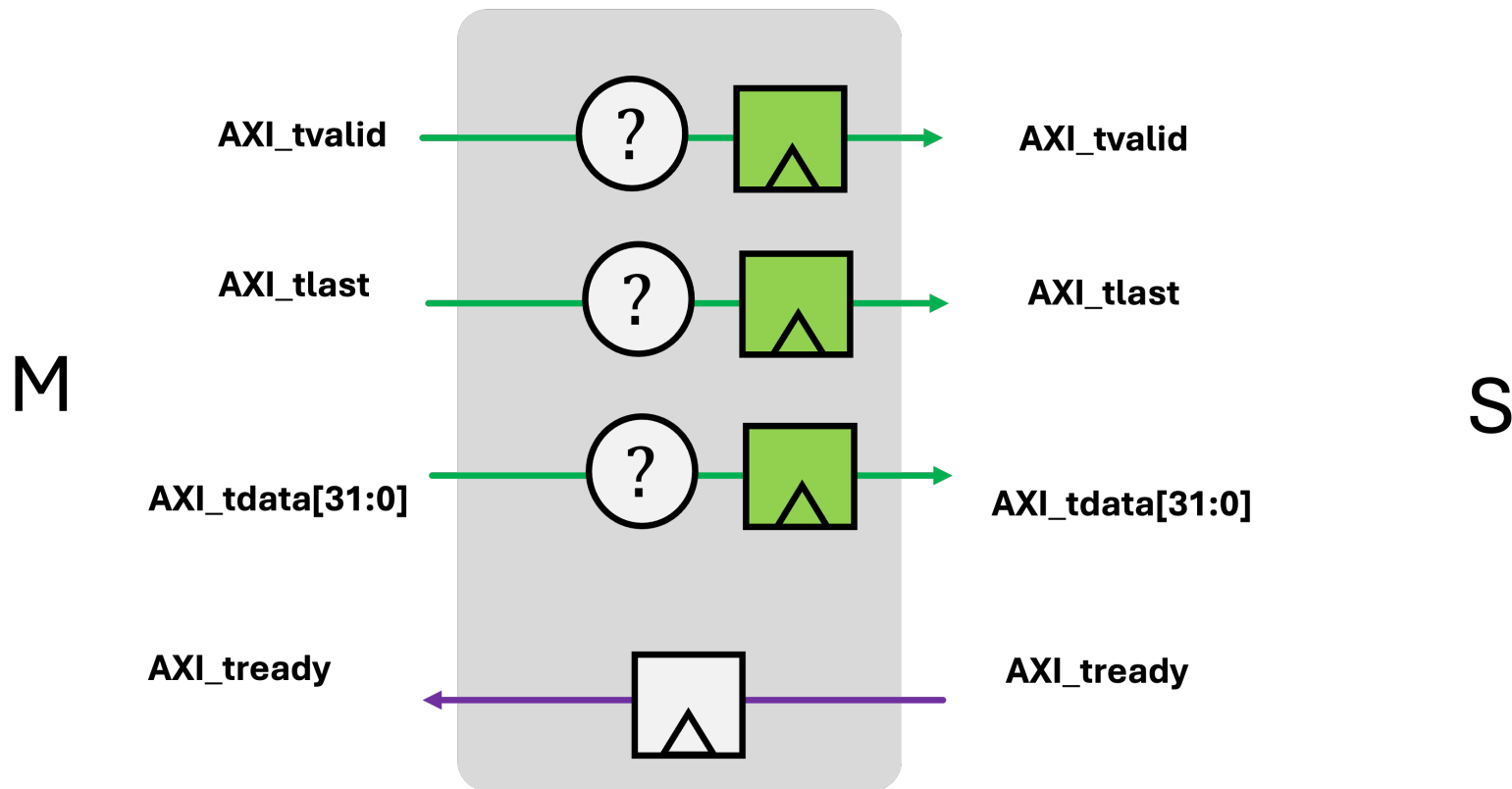
Nice Writeup

- Kind of an old-school FPGA writeup of a skid buffer found here:

https://fpgacpu.ca/fpga/Pipeline_Skid_Buffer.html

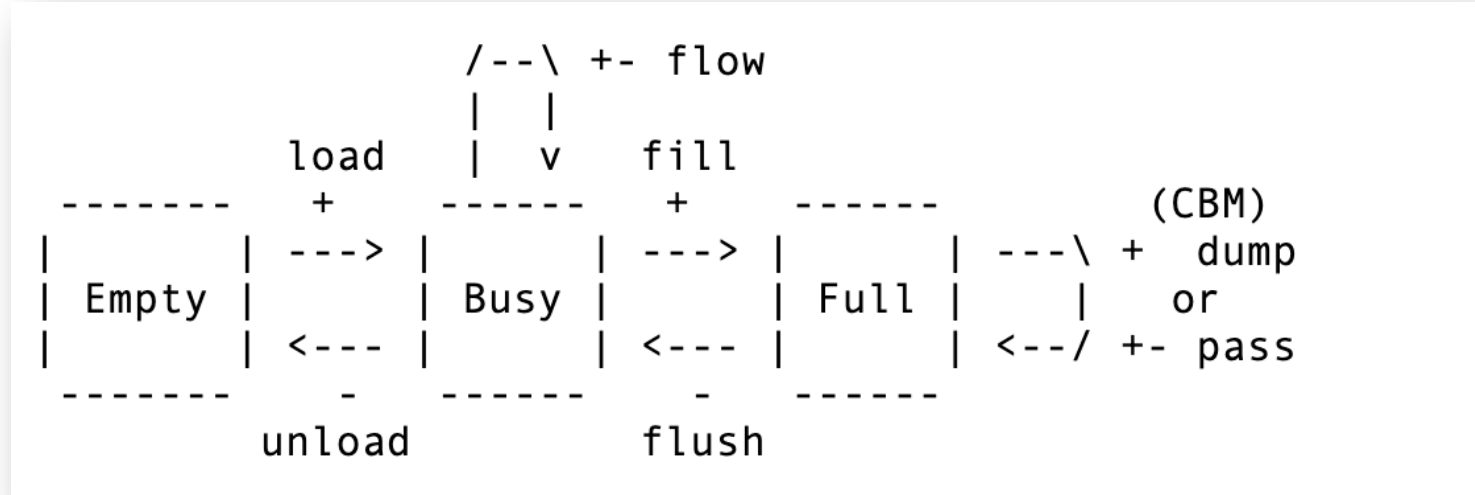
- You will write your own version this upcoming week.

Skid Buffer



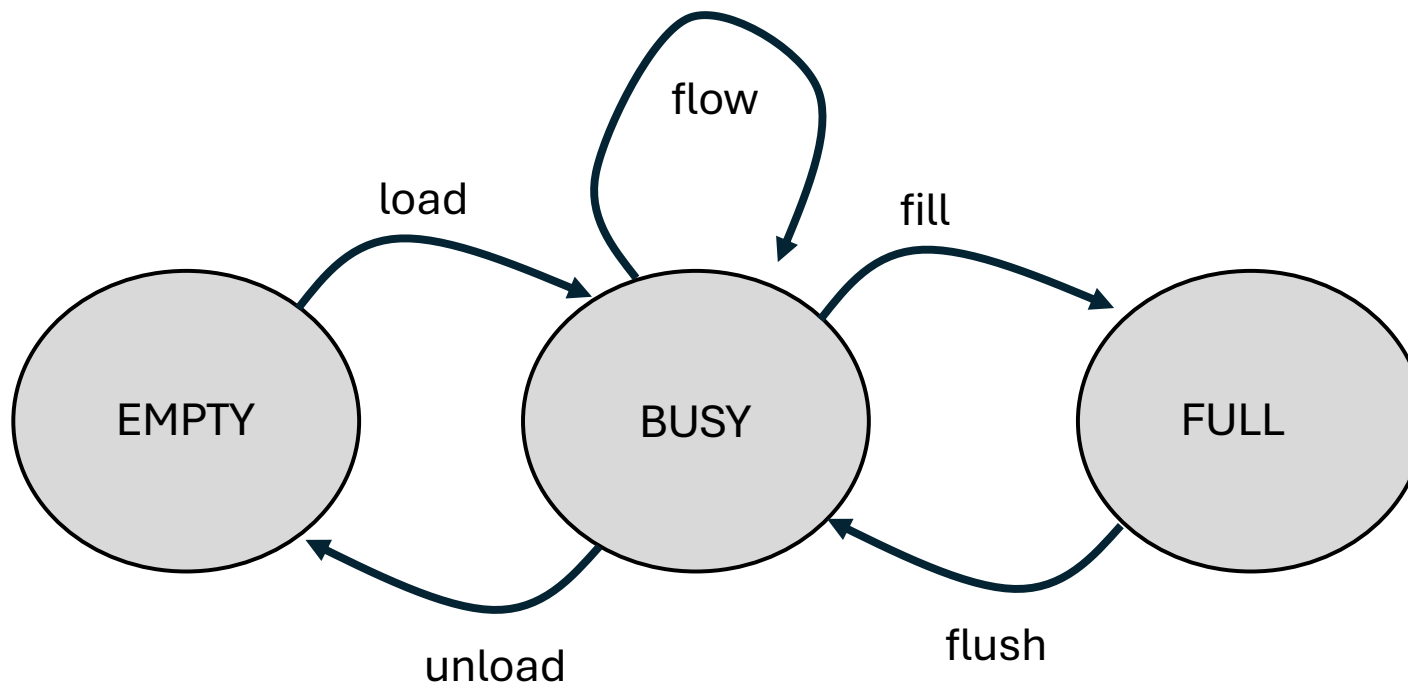
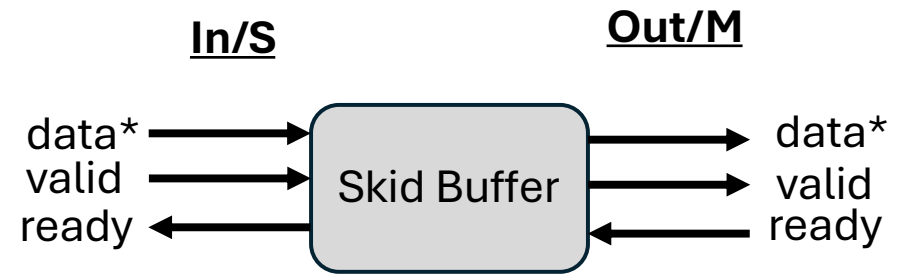
Simple FSM logic

- Three-state FSM can take care of this



https://fpgacpu.ca/fpga/Pipeline_Skid_Buffer.html

Example: Skid Buffer



Control Path

We separate the control path so the associated data path does not have to know anything about the current state or its encoding.

This FSM assumes the usual meaning and behaviour of valid/ready handshake signals: when both are high, data transfers at the end of the clock cycle. It is an error to raise ready when not able to accept data (thus losing the incoming data), or to raise valid when not able to send data (thus duplicating previously sent data). *These error situations are not handled.*

To operate our datapath as a skid buffer, we need to understand which states we want to allow it to be in, and which state transitions we also allow. This skid buffer has three states:

1. It is Empty.
2. It is Busy, holding one item of data in the main register, either waiting or actively transferring data through that register.
3. It is Full, holding data in both registers, and stopped until the main register is emptied and simultaneously refilled from the buffer register, so no data is lost or reordered. (Without an available empty register, the input interface cannot skid to a stop, so it must signal it is not ready.)
4. It is Full and in Circular Buffer Mode, holding data in both registers, and can accept new data into the buffer register while simultaneously replacing the contents of the main register with the current contents of the buffer register.

The operations which transition between these states are:

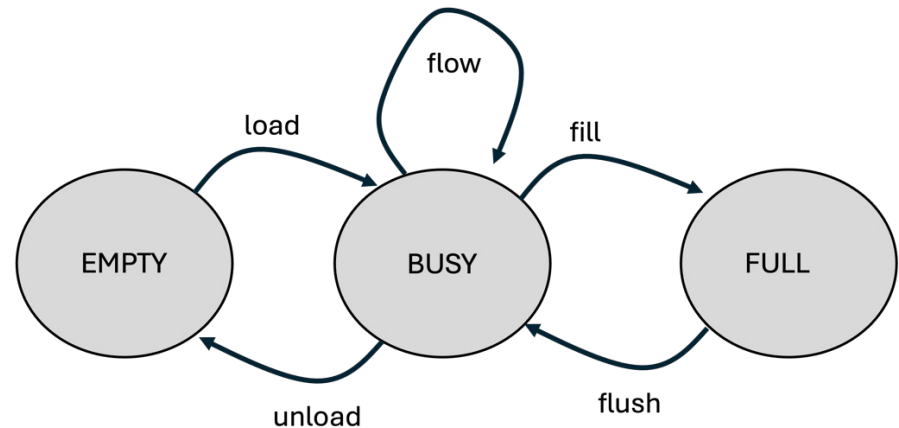
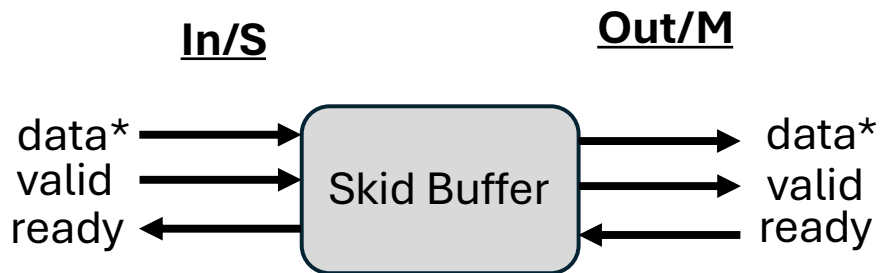
- the input interface inserting a data item into the datapath (+)
- the output interface removing a data item from the datapath (-)
- both interfaces inserting and removing at the same time (+-)

We also descriptively name each transition between states. These names will show up later in the code.



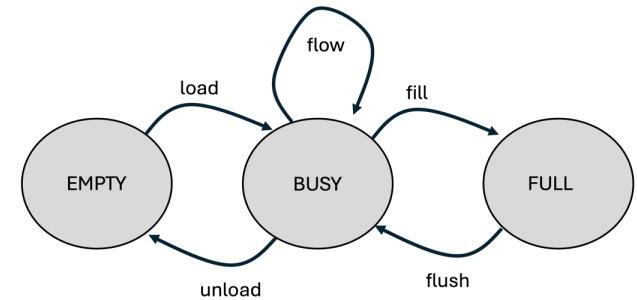
https://fpgacpu.ca/fpga/Pipeline_Skid_Buffer.html

Skid Buffer



- BUSY is normal operation where data is coming in and out.
- If there's a hiccup on the output side, go to FULL and stall pipeline (`s00_tready -> 0`)
- If there's a hiccup on the input side, go to EMPTY and stall pipeline (`m00_tvalid -> 0`)

Skid Buffer



- This simple FSM description...glosses over the potential complexity of the implementation: 3 states, each connected to 2 signals (valid/ready) per interface, for a total of 16 possible transitions out of each state, or 48 possible state transitions total.
- It is surprisingly difficult to write a skid buffer from scratch if you're not careful.
- We'll use the skid buffer as a platform to start looking into coverage next week.

Final Projects

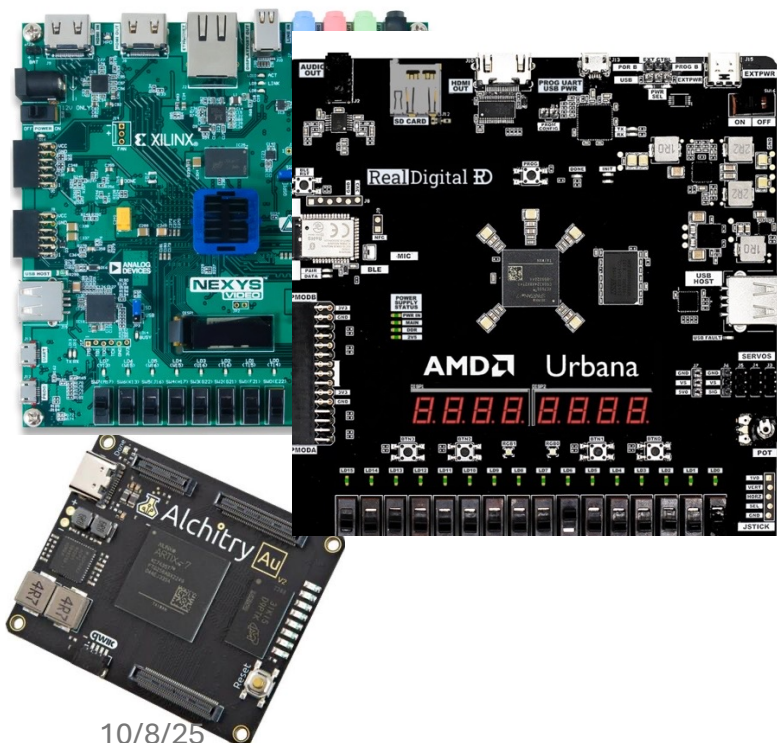
Final Projects

- After Week 8, we won't have labs. It'll be final project time.
- Projects will need to be done in teams (unless there is a very good reason to not do that)
- I'd like to start final projects on Nov 1.
- Everything is due Dec 10 at 5pm

Final Project Environment

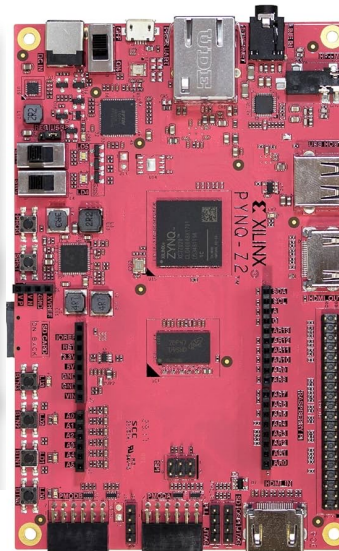
- Board Choice...don't really care, specifically...just pick one

Regular FPGAs



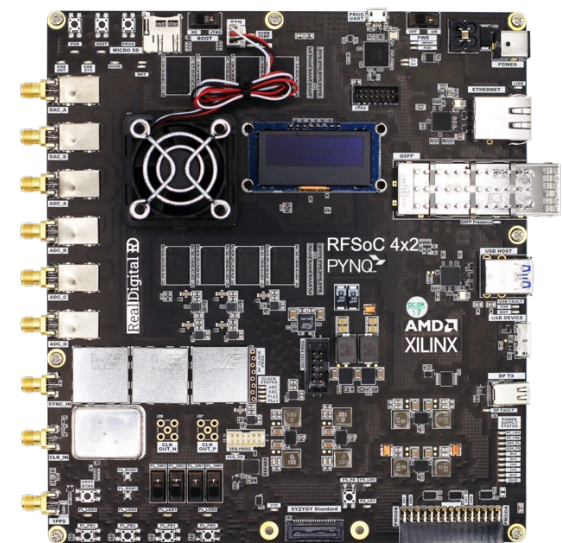
10/8/25

Standard SoCs



6.S965 Fall 2024

RFSoc

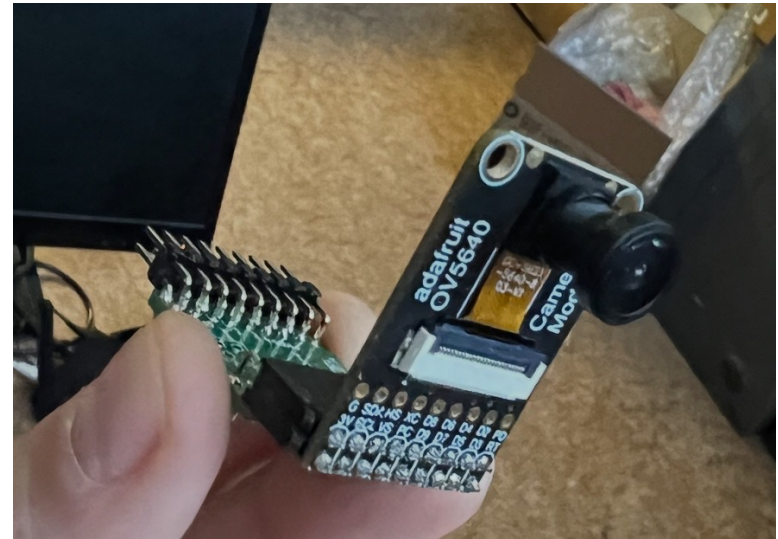


49

Regular FPGAs

- We have extra Urbana boards (and we have pretty well-functioning DDR interface logic this year!)
- Also a bunch of Nexys 4 DDR boards (about 2X resources of Urbana)
- Also have several Nexys Video boards (about 8X resources of Urbana)
- We also have those small Alchitry Au boards if anybody wants to do anything particularly mobile.

Additional Equipment



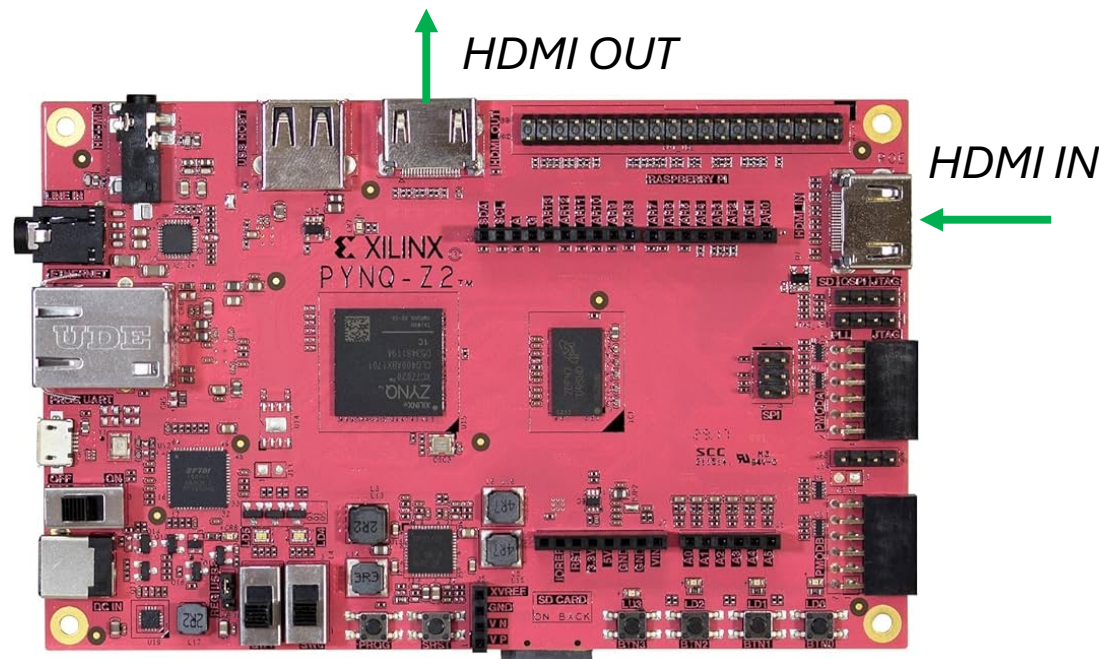
- We have those decent cameras from 6.205 now you can use
- Alternatively could use the HDMI input on the Pynq board if you're looking for a way to get easier/cleaner 720 video. I would also assume this would be easier to put test cases into

On that topic...Budget

- I have ~1.5K cash dollars or so to spend if we need to (for full class) so any antennas, amps, filters, we can get.

The Pynq Z2 Board

- Does have HDMI in and out, so it is possible to incorporate this into some interesting video-processing pipelines.



Anything with Computer Vision, this could be a good environment

- <https://www.youtube.com/watch?v=QPshQ9PsuFs>
- Sources of data could be computer feed (passthrough to monitor using HDMI in and out)

Project Idea: QICK

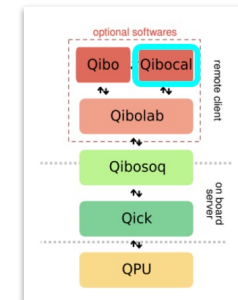
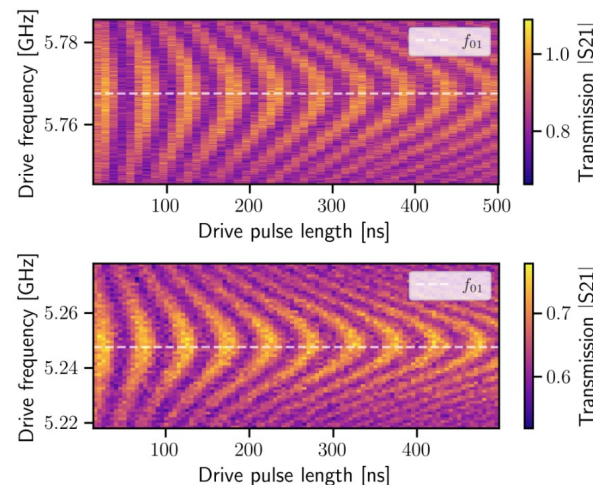
- I think quantum stuff is made up, but the RFSOCs are getting a lot of use in qubit analysis.
- Slides here:
 - https://docs.google.com/presentation/d/1rrgFXOn-ZROhKadeiPFLhmJAO-Ze_ezWnpoj1usnw/edit#slide=id.g30110b82c28_0_31
- I think there might be some interesting projects here involved in measuring these fake phenomena

QICK on a RFSoc

- Look at those Rabi oscillations (!) Very cool

Multi-qubit control with the Qibo/QICK stack

Simultaneous Rabi oscillations in neighboring qubits

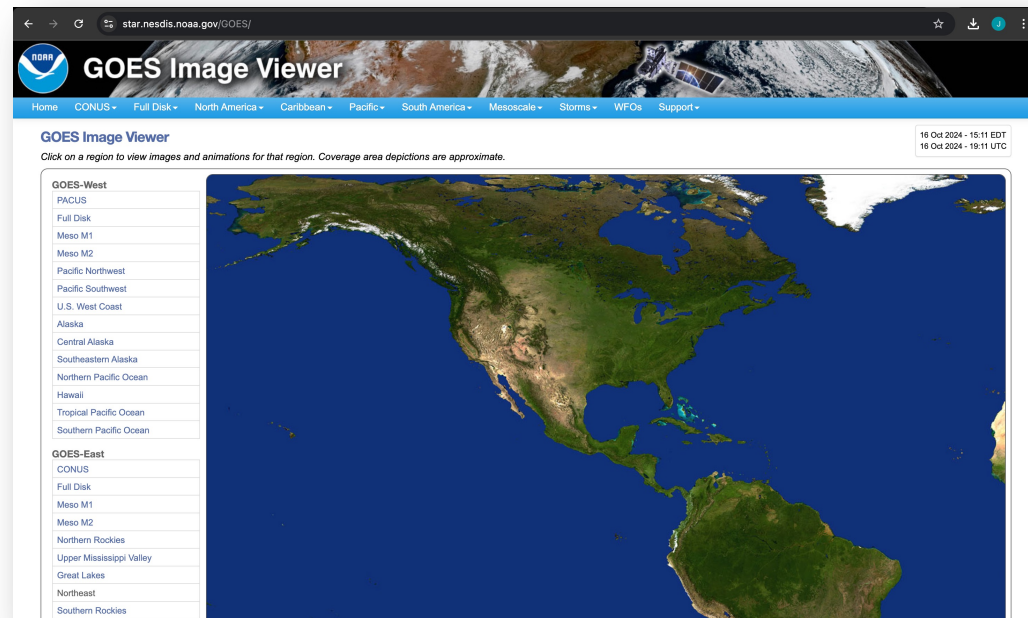


More by Qibo team:

- ❖ Characterization of 2 pairs of CZ interactions between 3 qubits
- ❖ Work-in-progress to use two synchronized ZCU216 QICK boards to control 8 qubits

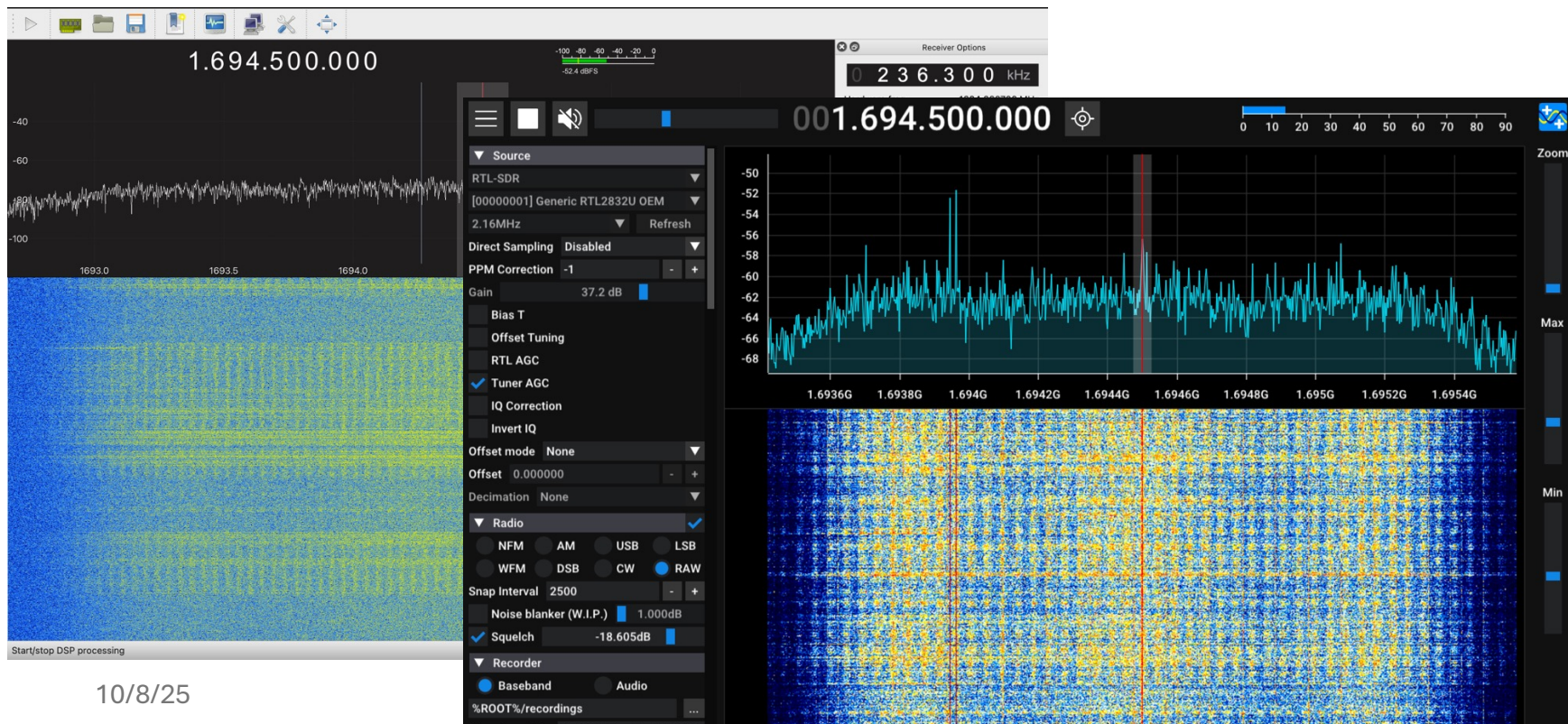
Satellites

- GOES (Geostationary Operational Environmental Satellites) send down really cool images with a complicated protocol
- Signals around 1.7 GHz



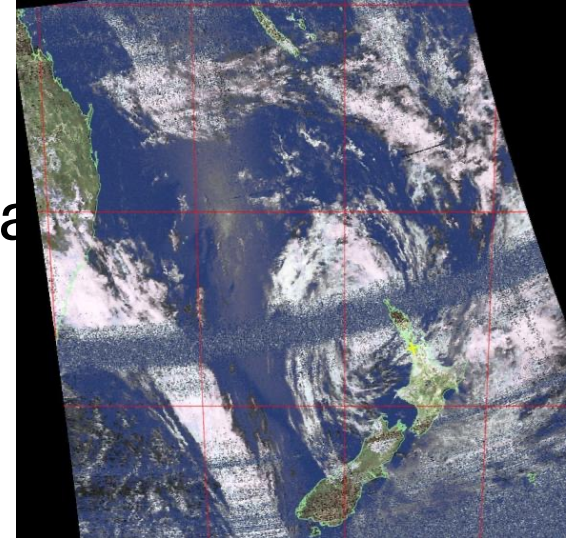
Satellites

- That dish I have on the 6th floor is meant for GOES...pointed it and with a front-end filter and LNA we were able to see some signal



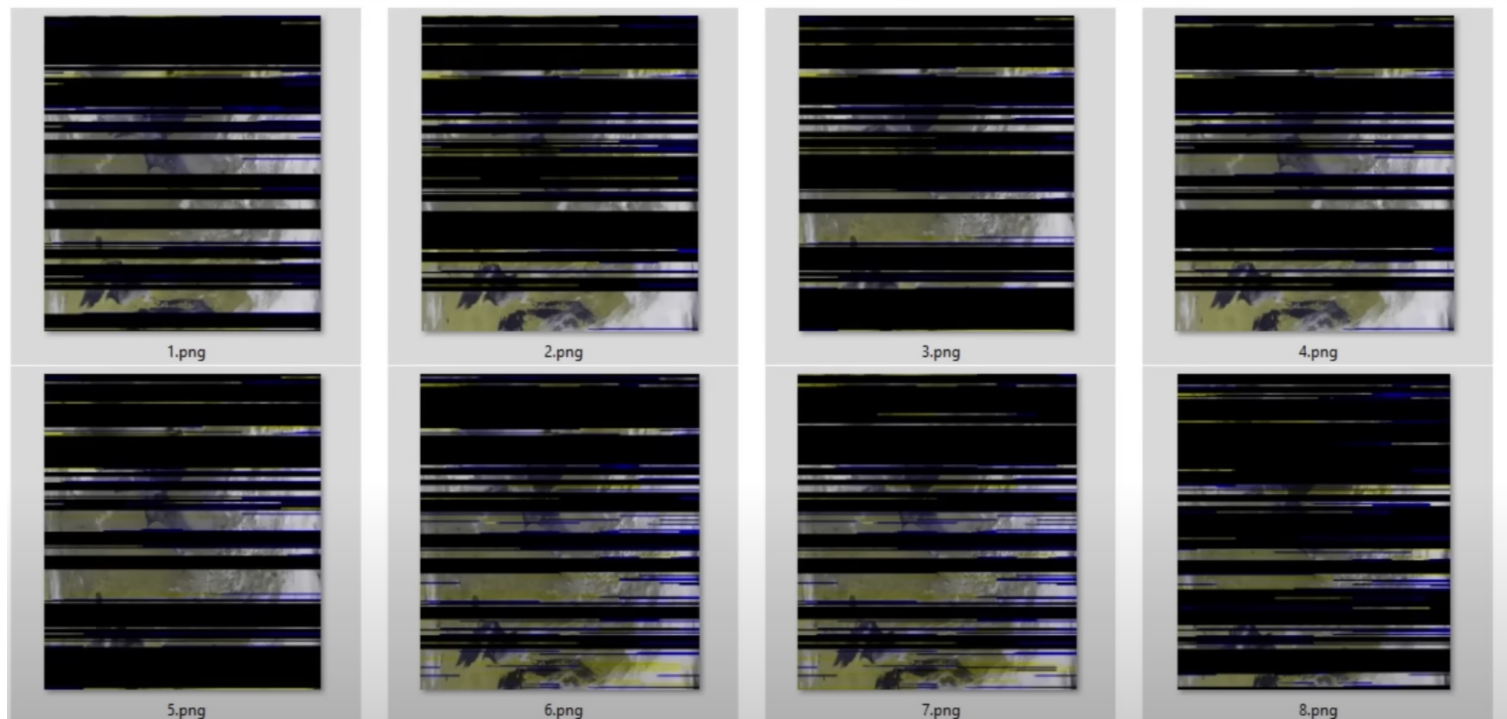
NOAA APT Satellites

- APT= Automatic Picture Transmission Satellites
- Not geostationary so will have to track...but should be possible...
- Send down
- Signals on/around 137 MHz
- Use circularly polarized antennas



Meteor Satellites (Russian)

- Russian weather satellites are also not geostationary but can give good images



International Space Station

- Periodically sends down SST (Slow-Scan Television) as well as other signals
- <https://www.youtube.com/watch?v=HaAprfh9ZtM>

Use some of these...parts

- Antennas
- Low Noise Amplifiers
- Some Front-end filters



Commercial FM radio

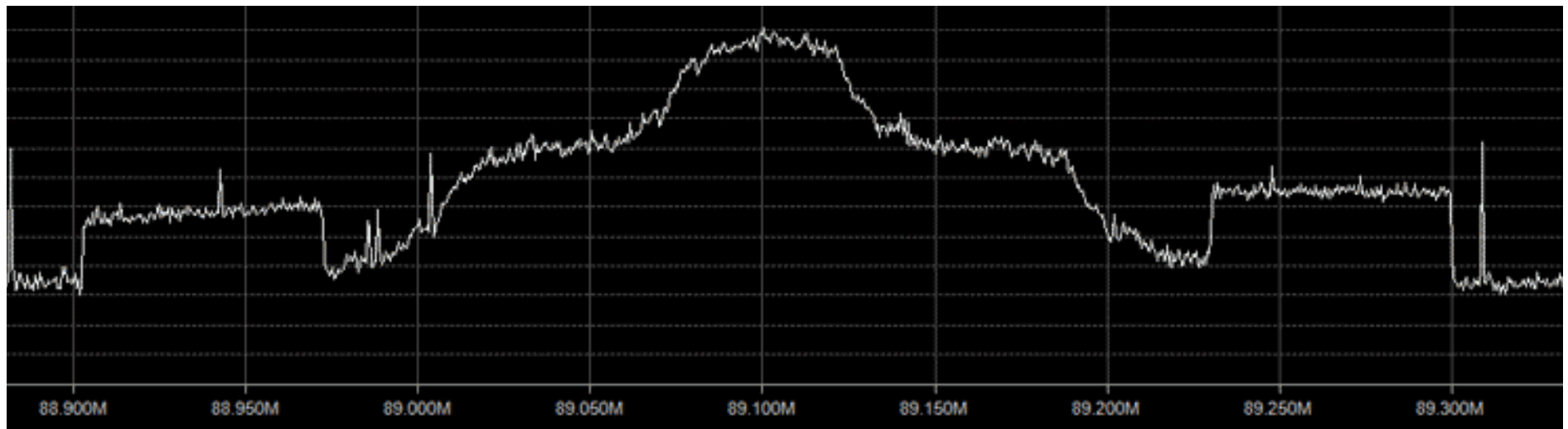


Traditional FM radio

- Just do regular FM demodulation (which you can do using IQ data). It isn't that easy.
- Make it tunable to listen to all the stations

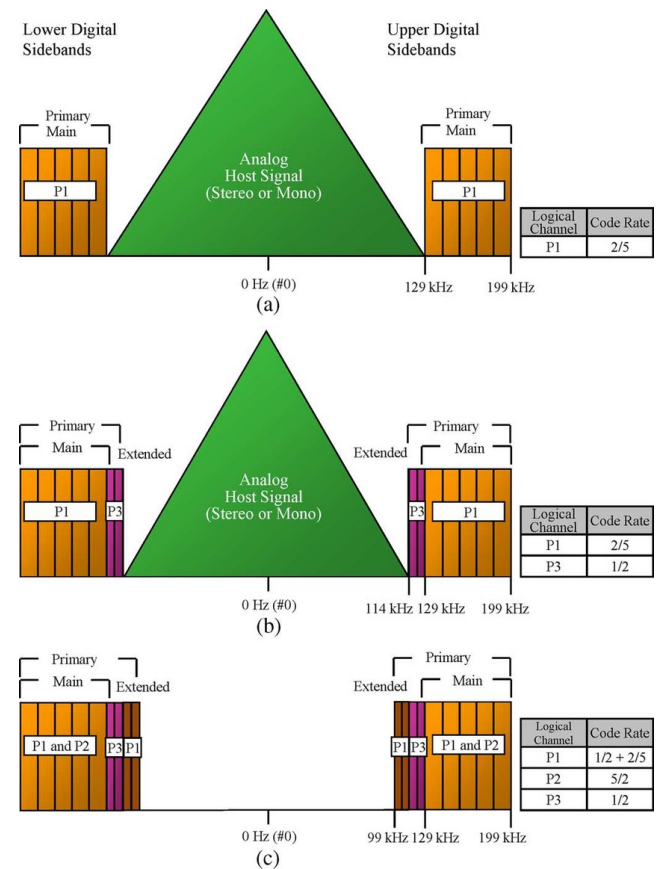
Digital Terrestrial Radio

- <https://www.rtl-sdr.com/decoding-and-listening-to-hd-radio>
- <https://github.com/theori-io/nrsc5-nrsc-5-with-an-rtl-sdr/>



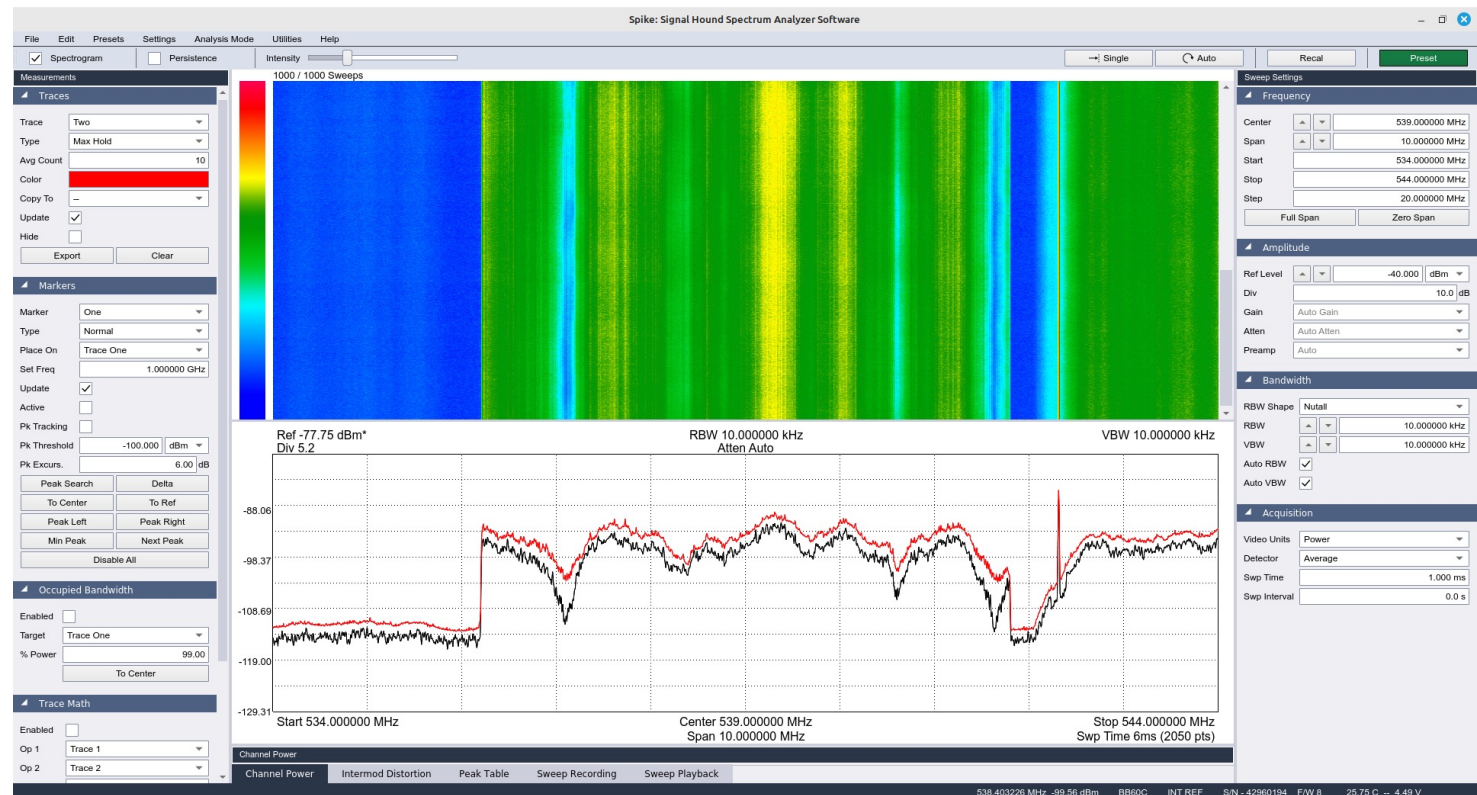
Digital Terrestrial Radio

- <https://www.rtl-sdr.com/decoding-and-listening-to-hd-radio>
- <https://github.com/theorio/nrsc5-nrsc-5-with-an-rtl-sdr/>



Digital Television

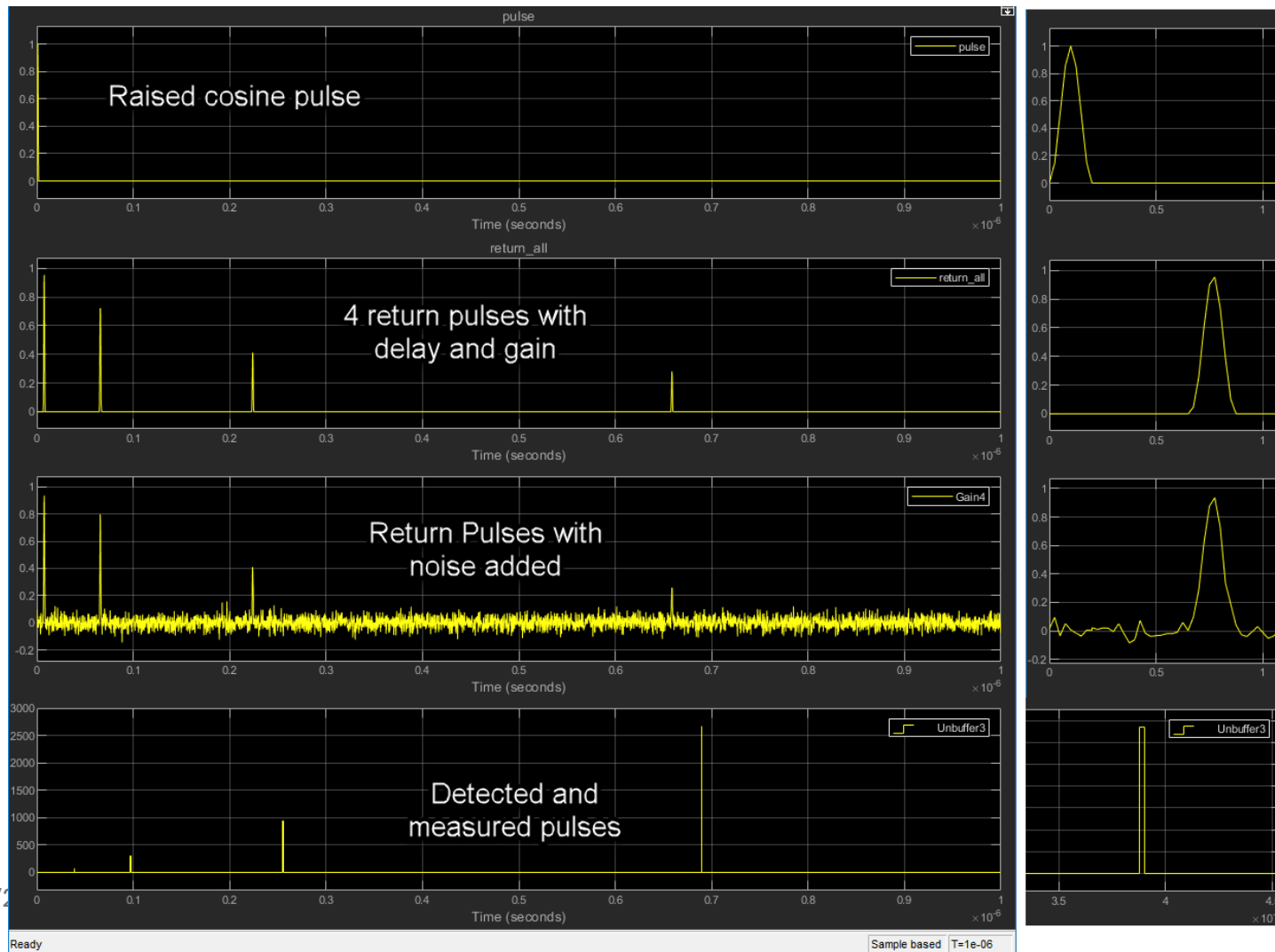
- Modern over-the air TV is digital. USA uses ATSC (Advanced Television Systems Committee)



LIDAR (use DAC)

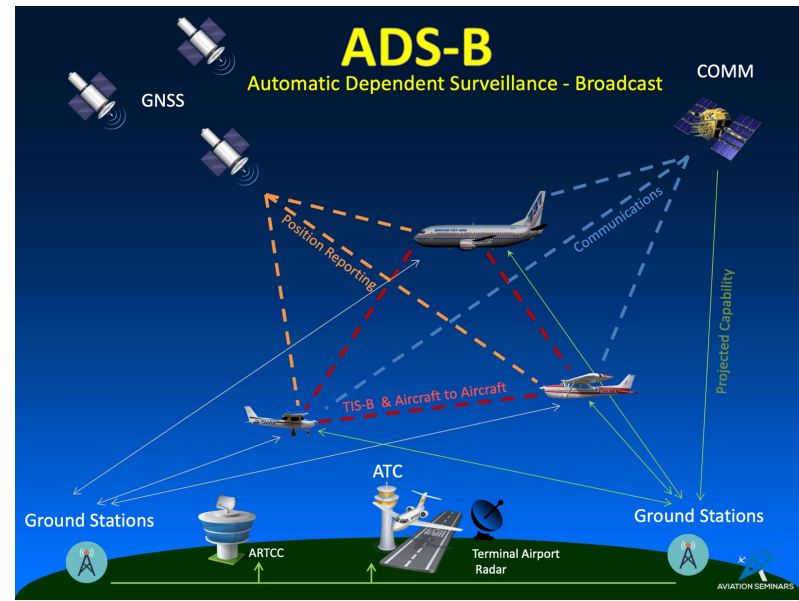
- Some sort of continuous wave LIDAR:
 - <https://www.bridgerphotonics.com/blog/frequency-modulated-continuous-wave-fmcw-lidar>
- AMD/Xilinx have this project here:
 - <https://www.xilinx.com/developer/articles/lidar-pulse-detection-accelerator-model-based-design-targeting-vitis-and-rfsoc.html>

Pulse-based LIDAR?



ADS-B

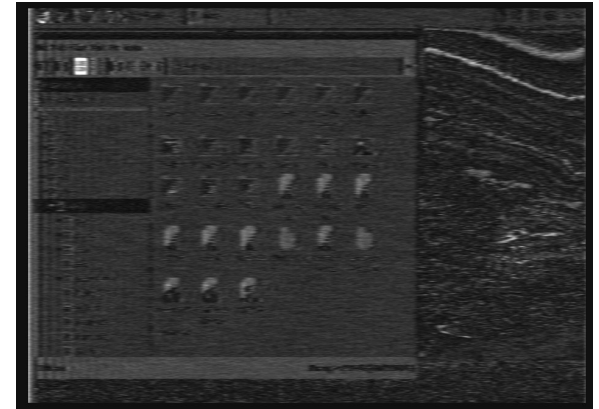
- I think this is going to be our lab



Some sort of minimal beam-targeting motion detection?

- Either use the DACs or use ambient signals to detect motion?

See HDMI from its noise?



<https://www.windytan.com/2023/02/using-hdmi-radio-interference-for-high.html>

Deep-TEMPEST: Using Deep Learning to Eavesdrop on HDMI from its Unintended Electromagnetic Emanations

Santiago Fernández
Emilio Martínez

sfernandez@fing.edu.uy
emartinez@fing.edu.uy

Facultad de Ingeniería, Universidad
de la República
Montevideo, Uruguay

Gabriel Varela

jorge.varela@fing.edu.uy

Facultad de Ingeniería, Universidad
de la República
Montevideo, Uruguay

Pablo Musé

Federico Larroca

pmuse@fing.edu.uy
flarroca@fing.edu.uy

Facultad de Ingeniería, Universidad
de la República
Montevideo, Uruguay

24

Abstract

Keywords

More recently 2024

<https://arxiv.org/pdf/2407.09717>

Old-style Television???

- Last year a team *almost* recreated analog television using the RFSoc
- So close
- Alternatively could read it in from like a video game system and convert to hdmi?



So...

- I will post a teaming form and a project idea submission thing this Friday/weekend
- I'll talk about final project ideas in any depth you want.